# Oracle HRMS

インプリメンテーション・ガイド（日本仕様）

リリース 11*i*

2002 年 5 月
部品番号：J06445-01

ORACLE®

Oracle HRMS インプリメンテーション・ガイド（日本仕様）, リリース 11*i*

部品番号：J06445-01

原本名：Implementing Oracle HRMS

原本部品番号：A73313-02

Copyright © 1996, 2000 Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されております。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation（米国オラクル）または日本オラクル株式会社（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

# 目次


## 第 I 部　導入

### 1　導入プラン

導入チェックリスト ..... 1-1
導入フローチャート ..... 1-2
インストール後のステップ ..... 1-17
DataInstall ..... 1-17
AutoPatch（adpatch） ..... 1-19
Quantum をインストールする ..... 1-20
マスター・データのロード ..... 1-22

### 2　導入ステップ

始める前に ..... 2-1
Oracle Applications Implementation Wizard ..... 2-2
管理 ..... 2-2
キー・フレックスフィールドを定義する ..... 2-2
付加フレックスフィールドを定義する ..... 2-14
その他情報タイプ（EIT）を定義する ..... 2-18
システム管理 ..... 2-19
アプリケーション・データ・エクスチェンジ（ADE）と階層ダイアグラマ ..... 2-20
ワーク・ストラクチャ ..... 2-21
組織体系を定義する ..... 2-21
役割を定義する ..... 2-24

等級関連情報を定義する ..... 2-26
給与情報を定義する ..... 2-28
報酬および福利厚生 ..... 2-29
入力パラメータ検証を定義する ..... 2-29
報酬および福利厚生を定義する ..... 2-31
給与のバランス、Formula および結果を定義する ..... 2-33
給与管理 ..... 2-34
不就業管理と有給休暇付与 (PTO) ..... 2-36
エレメント・セット ..... 2-40
総合報酬 ..... 2-41
「福利厚生」タブ・リージョン ..... 2-41
福利厚生の適格 ..... 2-41
導出適格要因 ..... 2-42
適格プロファイル ..... 2-44
ライフ・イベントを定義する（Oracle Advanced Benefits） ..... 2-44
プログラム設定 ..... 2-45
登録要件 ..... 2-47
アクティビティ・レート計算および補償範囲計算 ..... 2-48
レポート・グループ ..... 2-49
フレックス・クレジット計算（Oracle Advanced Benefits） ..... 2-50
管理 ..... 2-50
オンライン福利厚生サービスの定義 ..... 2-51
個人情報およびアサイメント ..... 2-52
Person タイプとアサイメント状況 ..... 2-52
特別個人情報（「個人特別情報」キー・フレックスフィールド体系） ..... 2-53
特定業務機能 ..... 2-56
予算計画 ..... 2-56
評価方式 ..... 2-57
要件照合 ..... 2-58
キャリア・マネジメントおよび後任管理 ..... 2-59
採用 ..... 2-59
キャリア・マネジメント ..... 2-59
査定および評価 ..... 2-62

**キャリア計画と後任計画** ..... 2-62

## 第 II 部　テクニカル・エッセイ

## 3　デートトラック

**デートトラックの機能** ..... 3-1
デートトラック・フォームの動作 ..... 3-1
デートトラック表の表体系 ..... 3-3
デートトラック表とビューの作成 ..... 3-4
フォーム・ユーザーが使用できるデートトラック・オプションの制限 ..... 3-6
**デートトラック履歴ビューの作成および修正** ..... 3-9
作成および変更できるものは？ ..... 3-9
デートトラック履歴を要求した場合に発生する処理 ..... 3-9
デートトラック履歴ビューの修正 / 変更に関するルール ..... 3-10
代替デートトラック履歴ビューの使用方法 ..... 3-12
デートトラック履歴ビューのリスト ..... 3-13

## 4　バッチ・エレメント・エントリ

**バッチ・エレメント・エントリのバッチ合計制御の作成処理** ..... 4-1
合計制御の設定 ..... 4-1
SQL コードの作成 ..... 4-1

## 5　給与処理

**概要** ..... 5-1
PYUGEN ..... 5-1
給与計算処理パラメータ ..... 5-3
給与処理の概要 ..... 5-3
アサイメント・レベル連動 ..... 5-4
**給与計算処理** ..... 5-5
アサイメントとエレメントの決定 ..... 5-5
各アサイメントの処理 ..... 5-6
実行結果と値の作成 ..... 5-8

コンテキストの設定 ..... 5-8
エレメント・スキップ・ルールの実行 ..... 5-9
バランスの作成と保守 ..... 5-9
Formula の実行 ..... 5-12
**支払前処理** ..... 5-16
支払方法の設定 ..... 5-16
現金支払の準備（UK、JP のみ） ..... 5-17
事前通知（US のみ） ..... 5-18
連結セット ..... 5-18
第三者支払（JP 以外） ..... 5-18
換算レート ..... 5-19
支払方法の上書き ..... 5-19
プロセス ..... 5-19
**支払処理** ..... 5-21
**磁気テープ処理** ..... 5-21
磁気テープ支払処理の実行 ..... 5-22
磁気テープ・レポートの実行 ..... 5-24
SRS 定義 ..... 5-25
磁気テープ処理の仕組み ..... 5-25
PL/SQL 駆動プロシージャ ..... 5-28
汎用 PL/SQL ..... 5-29
Formula インタフェース ..... 5-34
エラー処理 ..... 5-37
PL/SQL 例 ..... 5-37
**小切手ライター処理（JP 以外）** ..... 5-41
プロセス ..... 5-41
小切手番号付け ..... 5-44
小切手の無効化と再発行 ..... 5-45
再実行にマーク付け ..... 5-46
支払のロールバック ..... 5-46
SRW2 レポート ..... 5-46
PL/SQL プロシージャの使用および変更方法 ..... 5-48
**現金処理** ..... 5-49
**給与計算処理パラメータ** ..... 5-50
処理パラメータ値 ..... 5-50

処理パラメータの概要 ..... 5-50
パラレル処理パラメータ ..... 5-51
配列の選択、更新および挿入用のバッファ・サイズ・パラメータ ..... 5-52
原価計算固有のパラメータ ..... 5-54
磁気テープ固有のパラメータ ..... 5-54
エラー・レポート・パラメータ ..... 5-54
ロールバック固有のパラメータ ..... 5-55
給与計算処理のログ ..... 5-55
ロギング・パラメータ ..... 5-57
その他のパラメータ ..... 5-58
QuickPay 処理のシステム管理 ..... 5-59
**アサイメント・レベル連動** ..... 5-61
処理分類 ..... 5-61
ロールバック・ルールと再実行にマーク付けルール ..... 5-64
**GL への転送処理** ..... 5-66
**原価計算処理** ..... 5-67
給与の原価割当例 ..... 5-67
事業主費用の分配例 ..... 5-68

## 6 給与アーカイブ・レポータ（PAR）処理

**給与アーカイブ・レポータ（PAR）処理** ..... 6-1
PAR のモード ..... 6-1
PAR 処理の概要 ..... 6-2
設定ステップの概要 ..... 6-3
アーカイブ用のデータベース・アイテムの作成 ..... 6-4
Formula の記述 ..... 6-7
アサイメントおよびアサイメント処理のためのパッケージ・プロシージャの記述 ..... 6-7
PAR 処理の SRS 定義の提供 ..... 6-9
PAY_REPORT_FORMAT_MAPPINGS_F 表の行の挿入 ..... 6-10
例： INITIALIZATION_CODE および ARCHIVE_CODE ..... 6-12

## 7 Oracle Payroll のバランス

**Oracle Payroll のバランス** ..... 7-1
バランスの概要 ..... 7-1

最新バランス ..... 7-2
**バランス累計期間** ..... 7-4
**Oracle Payroll の初期化バランスのロード** ..... 7-8
概要 ..... 7-8
ステップ ..... 7-9
バランス・ロード処理 ..... 7-9
最新バランス ..... 7-11
初期化バランスに挿入するエレメントの設定 ..... 7-11
バランス初期値の設定 ..... 7-13
初期化バランスのアップロード処理 ..... 7-16
バランスの初期化ステップ ..... 7-21
**レポートへのバランス値の組込み** ..... 7-23
バランス関数 ..... 7-24
**国別バランス初期化（UK のみ）** ..... 7-26
バランス初期化エレメント ..... 7-26
**レポートへのバランス値の組込み（UK のみ）** ..... 7-31
バランス関数 ..... 7-31
**「バランスの表示」の使用方法（UK のみ）** ..... 7-33

## 8 給与明細レポート（UK のみ）

**給与明細レポート（UK のみ）** ..... 8-1
パラメータ値 ..... 8-1
問合せ ..... 8-2
グループ ..... 8-2
トリガー ..... 8-3
レイアウト ..... 8-3
動的ソート順 ..... 8-3

## 9 FastFormula

**FastFormula アプリケーション・ディクショナリ** ..... 9-1
ディクショナリ内のエンティティ ..... 9-1
新規データベース・アイテムの定義 ..... 9-4
**PL/SQL からの FastFormula のコール** ..... 9-13
実行エンジン・インタフェース ..... 9-13
R11i での変更点 ..... 9-15

サーバー側インタフェース ..... 9-15
クライアント側のコール・インタフェース ..... 9-20
特別なフォーム・コール・インタフェース ..... 9-24
ロギング・オプション ..... 9-26

## 10 フレックスフィールド値の検証

**フレックスフィールド値の検証** ..... 10-1
ユーザー・プロファイル・オプションの参照 ..... 10-2
フォーム・ブロック・フィールド項目の参照 ..... 10-4
FND_SESSIONS 行の参照 ..... 10-5
不完全なコンテキスト・フィールド値リスト ..... 10-6

## 11 オラクル人事管理システムにおけるセキュリティの拡張

**オラクル人事管理システムにおけるセキュリティの拡張** ..... 11-1
セキュリティ・プロファイル ..... 11-1
セキュリティ処理 ..... 11-8
カスタム表のセキュリティ保護 ..... 11-11

## 12 オラクル人事管理システムの API

**オラクル人事管理システムの API** ..... 12-1
API の概要 ..... 12-3
オブジェクト・バージョン番号（OVN）について ..... 12-5
API パラメータ ..... 12-7
API 機能 ..... 12-22
API のフレックスフィールド ..... 12-23
複数言語サポート ..... 12-24
代替 API ..... 12-25
API エラーと警告 ..... 12-26
PL/SQL バッチ・プログラム例 ..... 12-28
WHO 列と Oracle Alert ..... 12-31
API ユーザー・フック ..... 12-32
基本ブロックとしての API の使用方法 ..... 12-54
Oracle Forms におけるオブジェクト・バージョン番号の処理方法 : ..... 12-55

## 13 Oracle HRMS Data Pump


**Oracle HRMS Data Pump** ..... 13-1
概要 ..... 13-2
Data Pump の使用方法 ..... 13-6
メタマッパーの実行 ..... 13-8
バッチ表へのデータのロード ..... 13-14
Data Pump プロセスの実行 ..... 13-17
エラーの検索と修正 ..... 13-19
データのパージ ..... 13-23
サンプル・コード ..... 13-24
生成済インタフェースの使用方法に関する注意 ..... 13-27
Data Pump で使用可能なユーティリティ・プロシージャ ..... 13-31
表およびビューの説明 ..... 13-32


## 14 Oracle Generic Third Party Payroll Backfeed


**Oracle Generic Third Party Payroll Backfeed（オラクル以外の給与システムのバックフィード）** ..... 14-1
**概要** ..... 14-2
**Oracle Generic Third Party Payroll Backfeed の設定** ..... 14-3
Oracle Generic Third Party Payroll Backfeed のインストール ..... 14-4
支払情報 ..... 14-6
バランス・タイプ ..... 14-6
API ..... 14-8
Data Pump の設定 ..... 14-8
使用するアップロード・オプションの決定 ..... 14-10
データ・アップローダの設定 ..... 14-10
**バックフィードを使用して給与計算結果をアップロードする方法** ..... 14-14
アップロード・ワークブックの作成 ..... 14-15
Load Sheets マクロの使用 ..... 14-16
Save Sheets マクロの使用 ..... 14-17
データ・アップローダの実行 ..... 14-18
Data Pump の実行 ..... 14-18
**Oracle 以外の給与システムの計算結果のオラクル人事管理システムでの表示** ..... 14-19
**参照情報** ..... 14-21
バックフィード用の Data Pump 関数 ..... 14-21
表の説明 ..... 14-23

PER_BF_BALANCE_AMOUNTS ..... 14-23
PER_BF_BALANCE_TYPES ..... 14-24
PER_BF_PAYMENT_DETAILS ..... 14-25
PER_BF_PAYROLL_RUNS ..... 14-26
PER_BF_PROCESSED_ASSIGNMENTS ..... 14-27
Data Pump が生成するパッケージおよびビュー ..... 14-28
バックフィードにより使用されるビジネス・プロセス API ..... 14-36

## 15 SQL トレース

**SQL トレース** ..... 15-1
SQL トレースの使用 ..... 15-2
SQL トレースの有効化 ..... 15-3
トレース・ファイルの検索 ..... 15-7
TKPROF とは? ..... 15-8
TKPROF を使用したトレース・ファイルの書式設定 ..... 15-9
TKPROF のソート・オプション ..... 15-11
TKPROF レポートについて ..... 15-12
生成された SQL トレース・ファイルの例 ..... 15-19
イベント 10046 を使用した詳細な SQL トレース ..... 15-20

## 用語集

# 第Ⅰ部

## 導入

# 1

# 導入プラン

## 導入チェックリスト

次のチェックリストを使用して、使用するオラクル人事管理システムを記録します。次に、導入する業務機能に関して実行する必要がある手順の概要を導入フローチャートで確認します。

> **注意：**オラクル人事管理システムを導入する前に必要なステップについては、1-1 ページの「導入プラン」を参照してください。

- ❑ 「導入ステップ」(2-1)（必須）

キー・フレックスフィールドおよび付加フレックスフィールド、その他情報タイプ（EIT）、通貨、" 全表示 " 人事管理システム・ユーザー、参照、アプリケーション・データ・エクスチェンジ（ADE）が含まれます。

- ❑ 「導入ステップ」(2-1)（必須）

組織、役職、職階、等級、給与が含まれます。

- ❑ 「導入ステップ」(2-1)（オプション）

報酬エレメント、入力パラメータ検証、バランス、Formula、給与管理、不就業管理 / 有給休暇付与、エレメント・セットが含まれます。

- ❑ 「導入ステップ」(2-1)（オプション）

オンライン福利厚生サービス、福利厚生適格、適格要因、ライフ・イベント、プログラム設定、フレックス・クレジット計算が含まれます。

- ❑ 「導入ステップ」(2-1)（必須）

Person タイプ、アサイメント状況、特別個人情報が含まれます。

- ❑ 「導入ステップ」(2-1)（オプション）

要員計画、評価方式、要件照合が含まれます。

- 「導入ステップ」(2-1)（オプション）

採用、キャリア・マネジメント、評価、後任計画が含まれます。

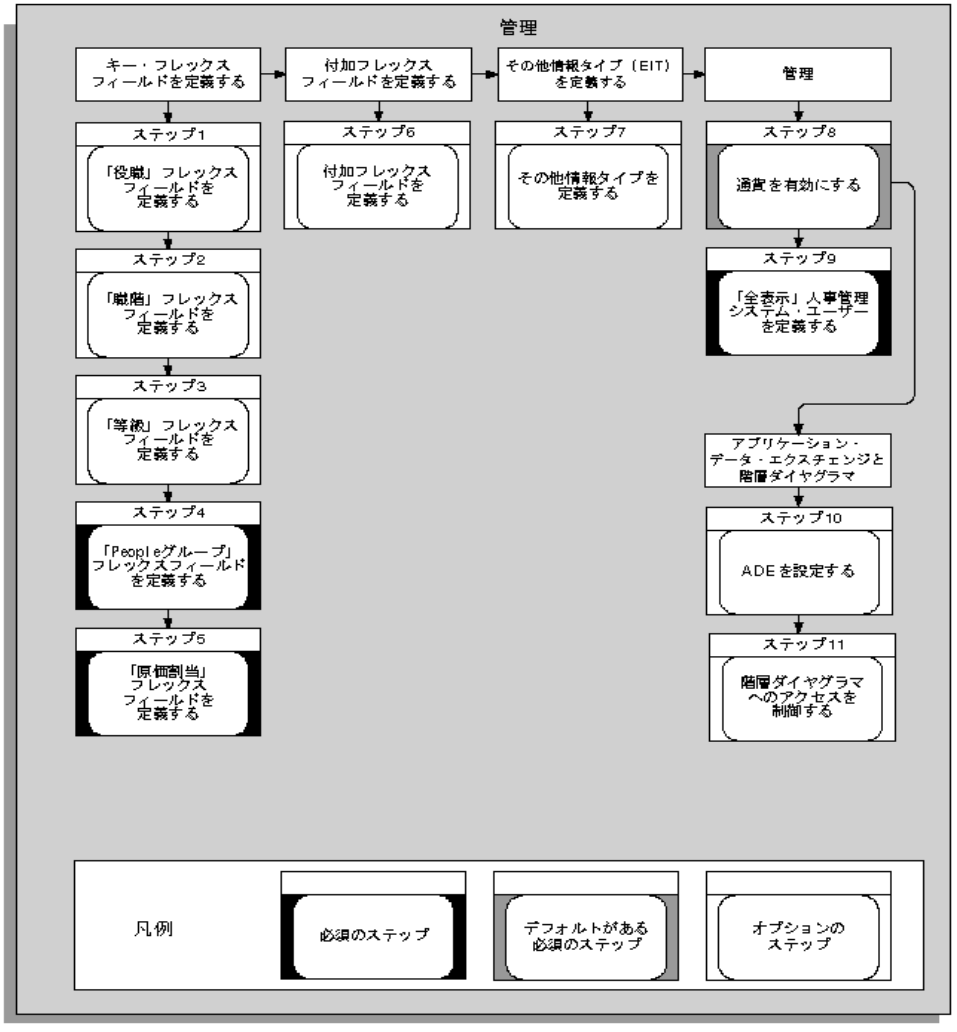
- 「制御」（オプション）

レポート、レター生成、カスタマイゼーション、タスク・フロー、ユーザー・セキュリティ、監査証跡、iHelp が含まれます。

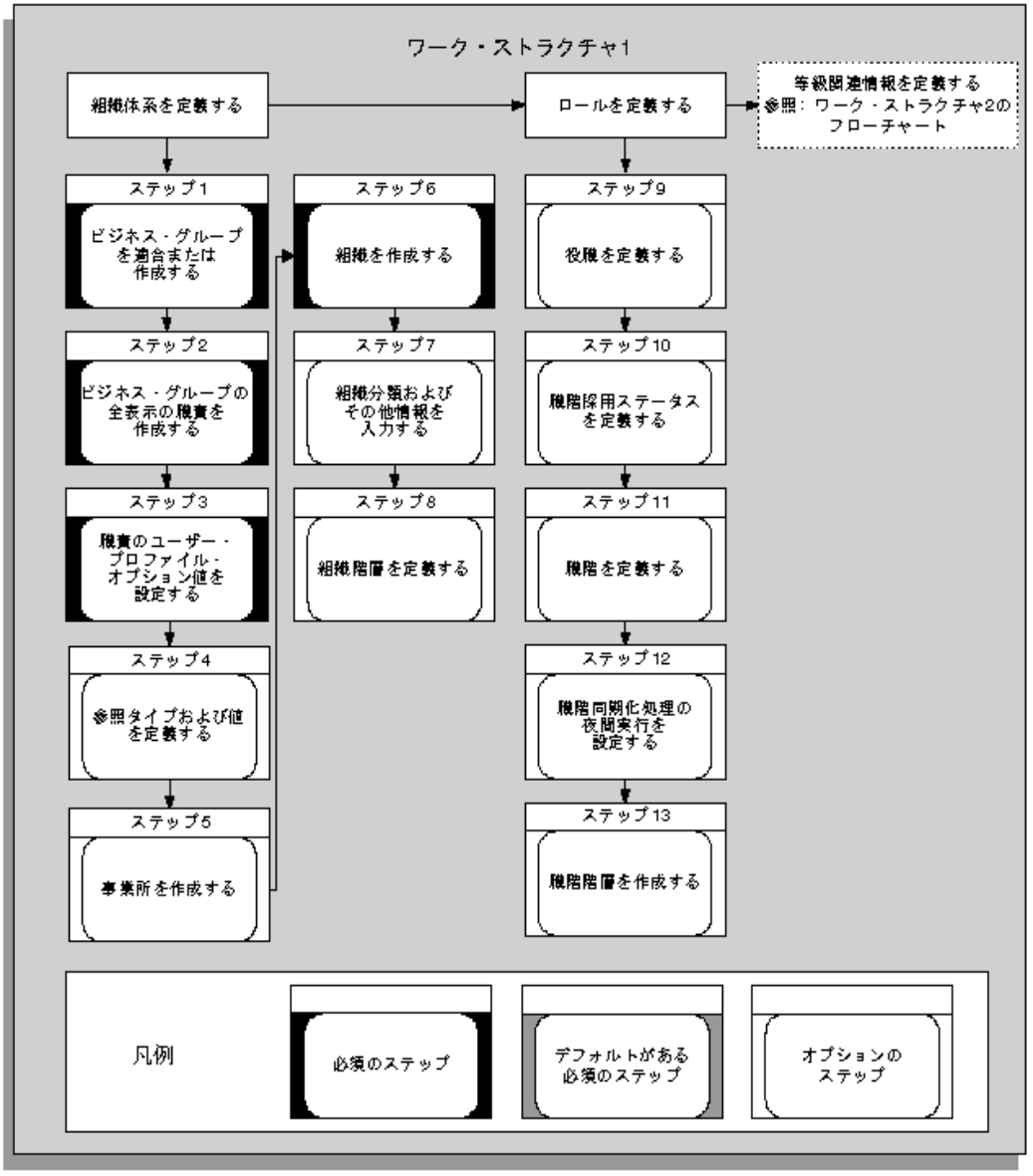
# 導入フローチャート

このセクションで説明する手順には、必須のものとオプションのものがあります。「デフォルトがある必須」とは、設定機能のデータベースに事前定義済みのデフォルト値が用意されていることを意味します。ただし、これらのデフォルト値は、レビューした上でビジネス要件に合わせて変更するかどうか判断する必要があります。デフォルト値を変更する必要がある場合は、該当する設定ステップを実行します。オプションのステップは、関連する機能を使用したり、特定業務機能を完了したりする場合に使用します。

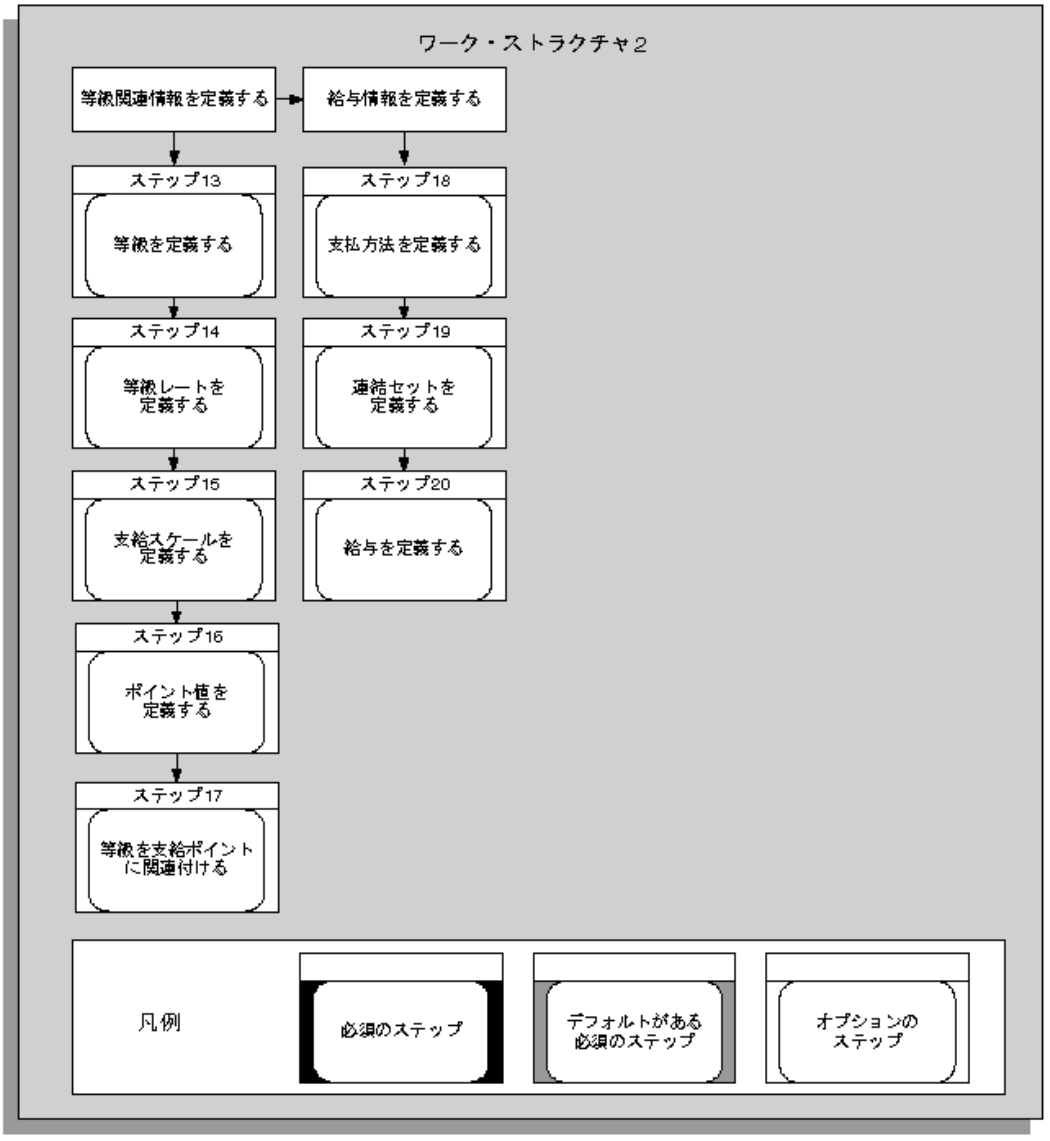
## 導入フローチャート : 管理



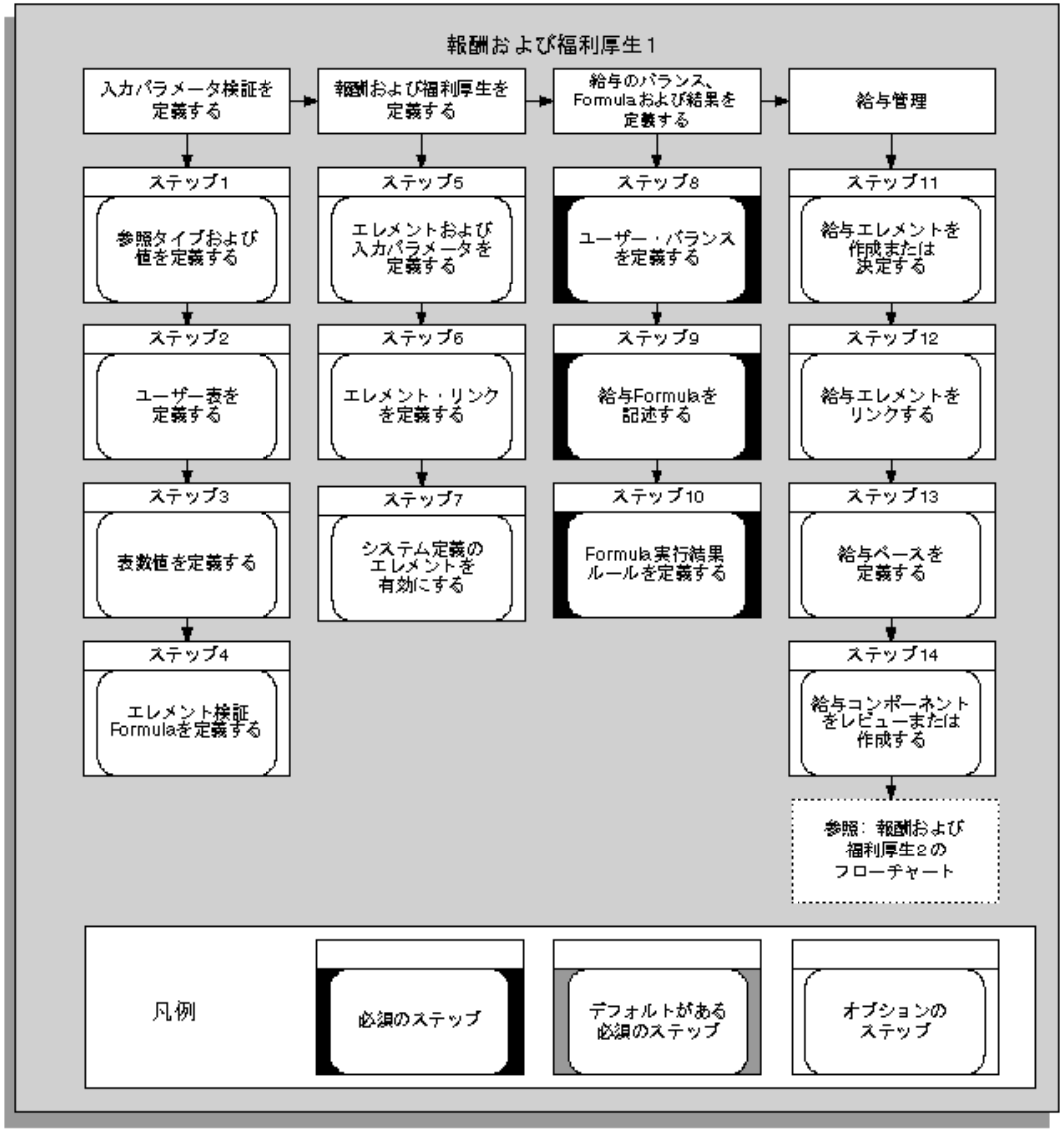
**導入フローチャート : ワーク・ストラクチャ 1**



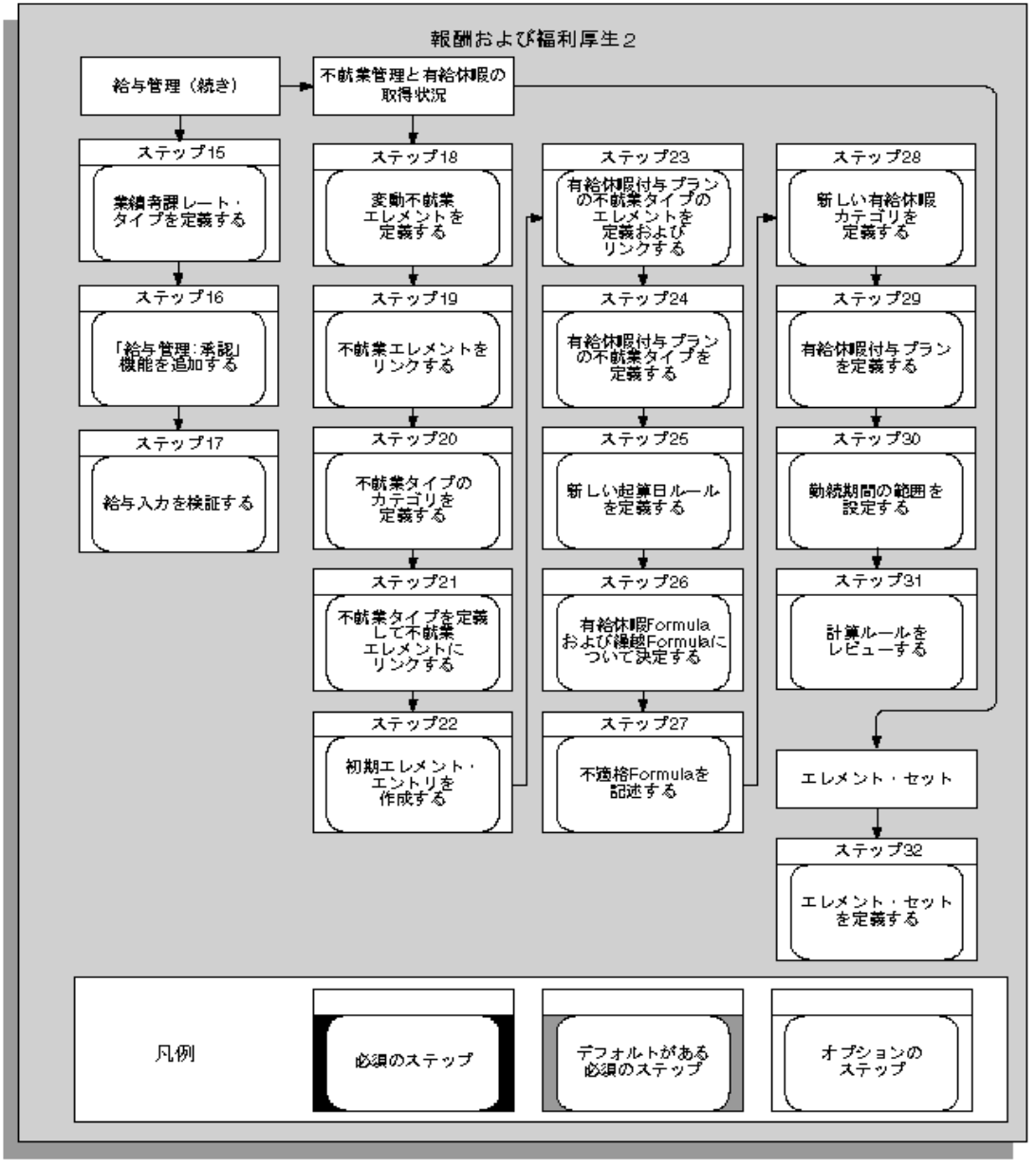
導入フローチャート : ワーク・ストラクチャ 2



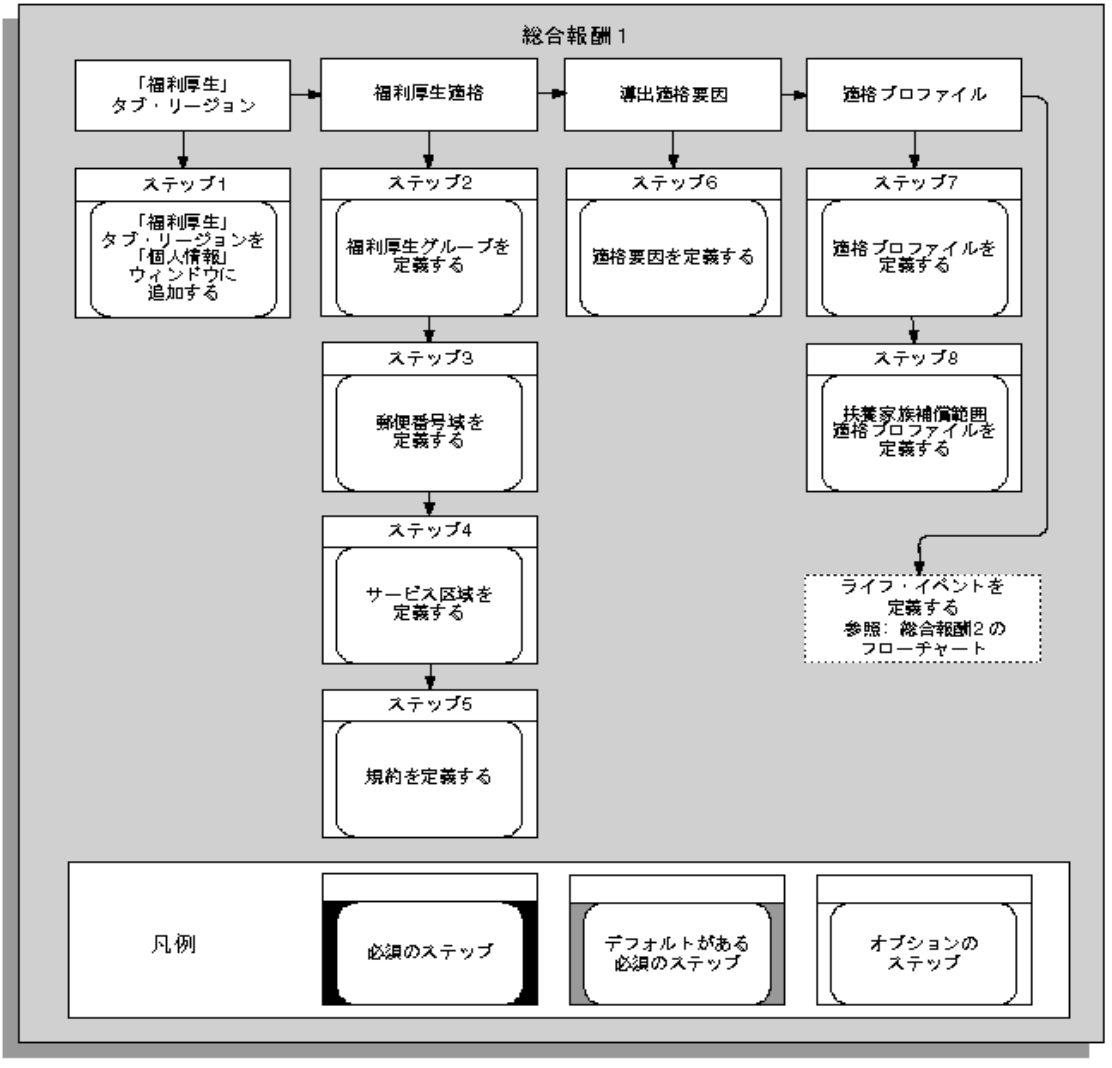
### 導入フローチャート : 報酬および福利厚生 1



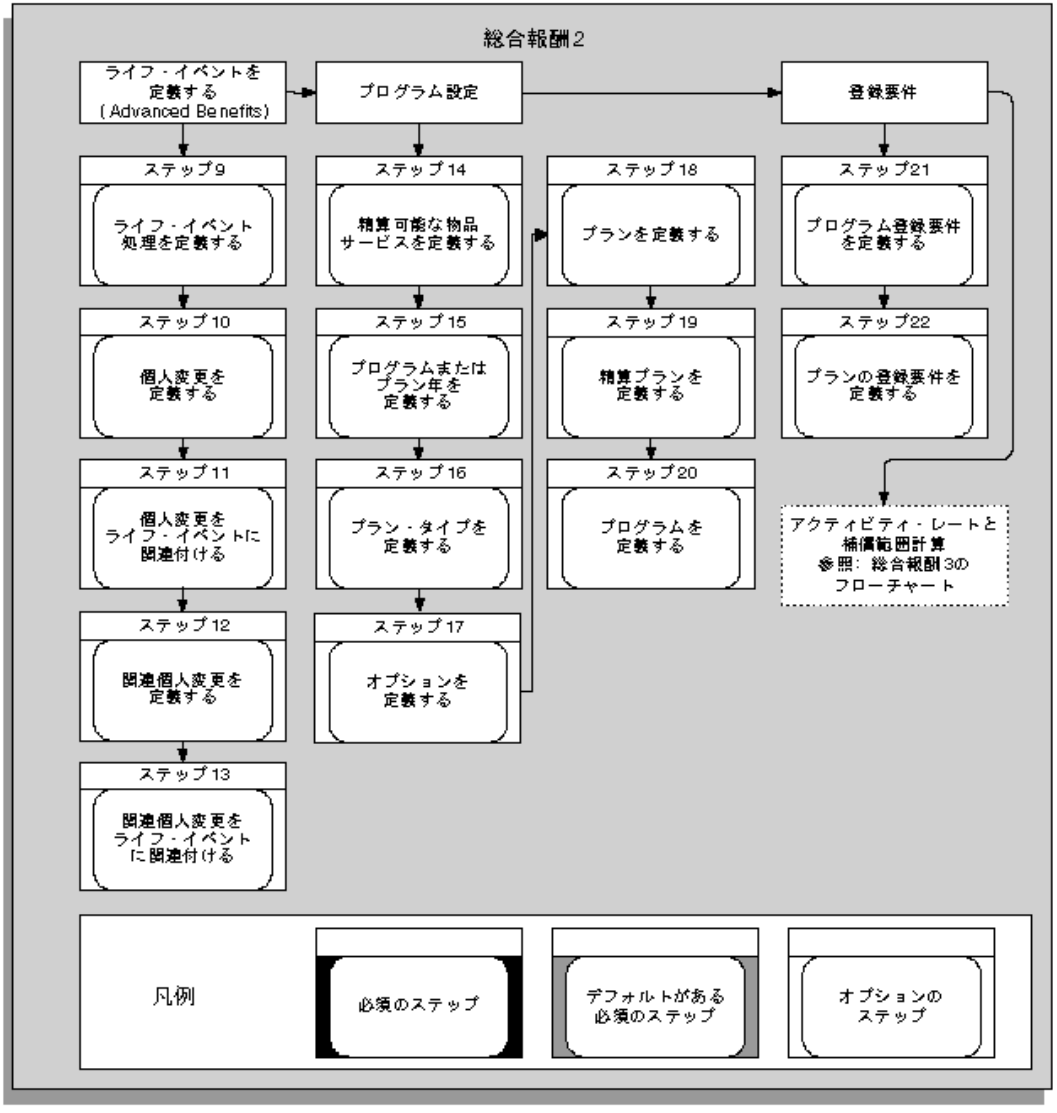
**導入フローチャート : 報酬および福利厚生 2**



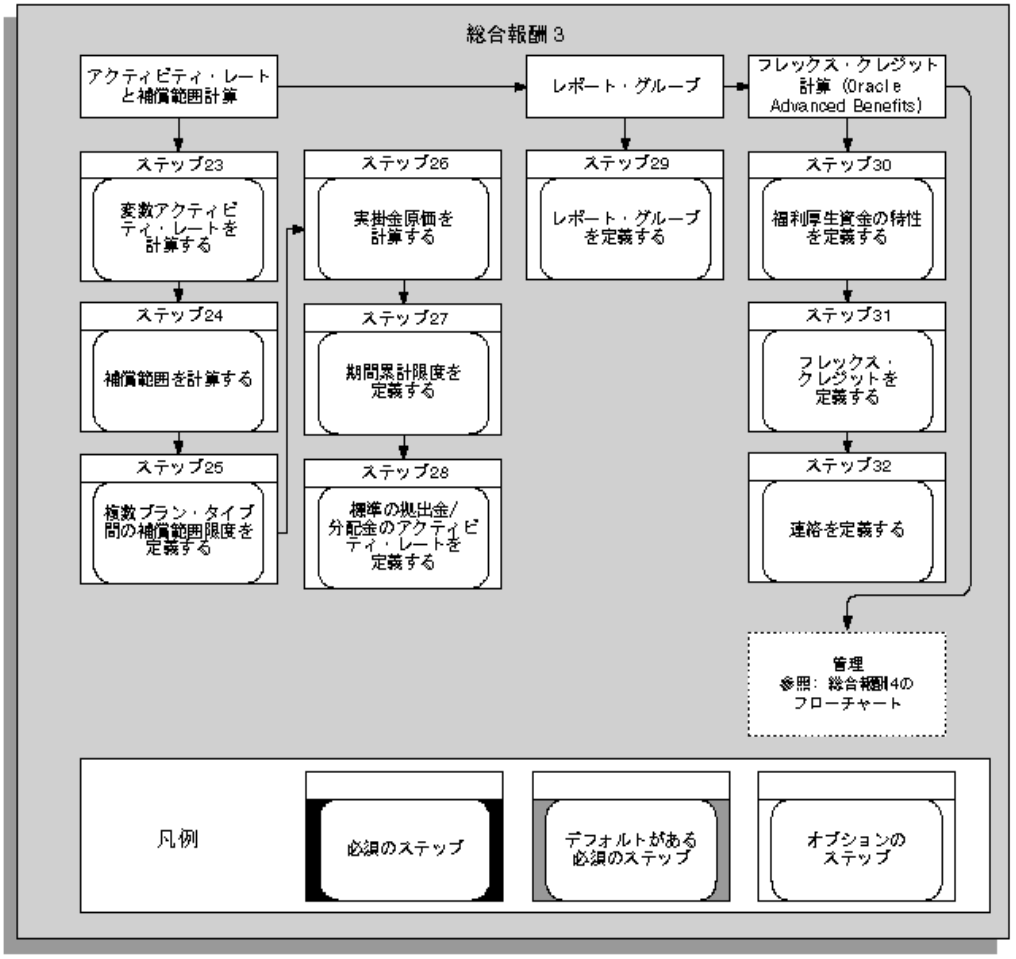
**導入フローチャート : 総合報酬 1**



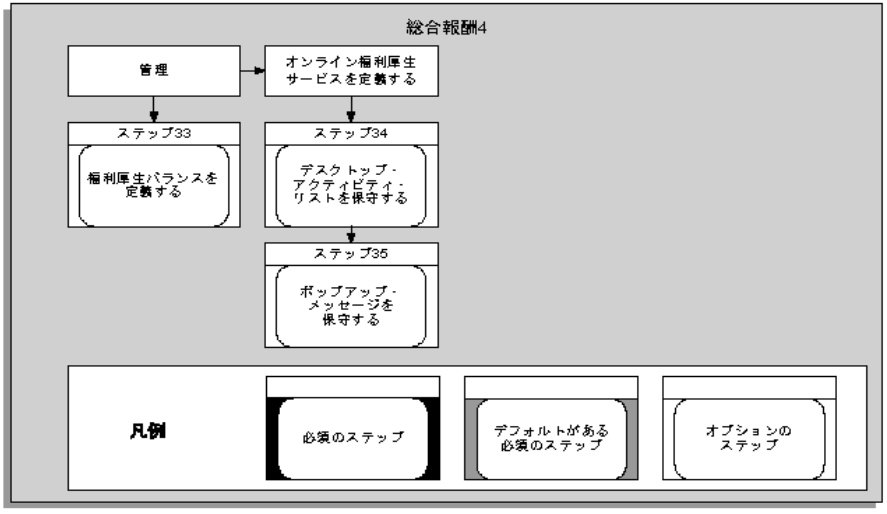
**導入フローチャート : 総合報酬 2**



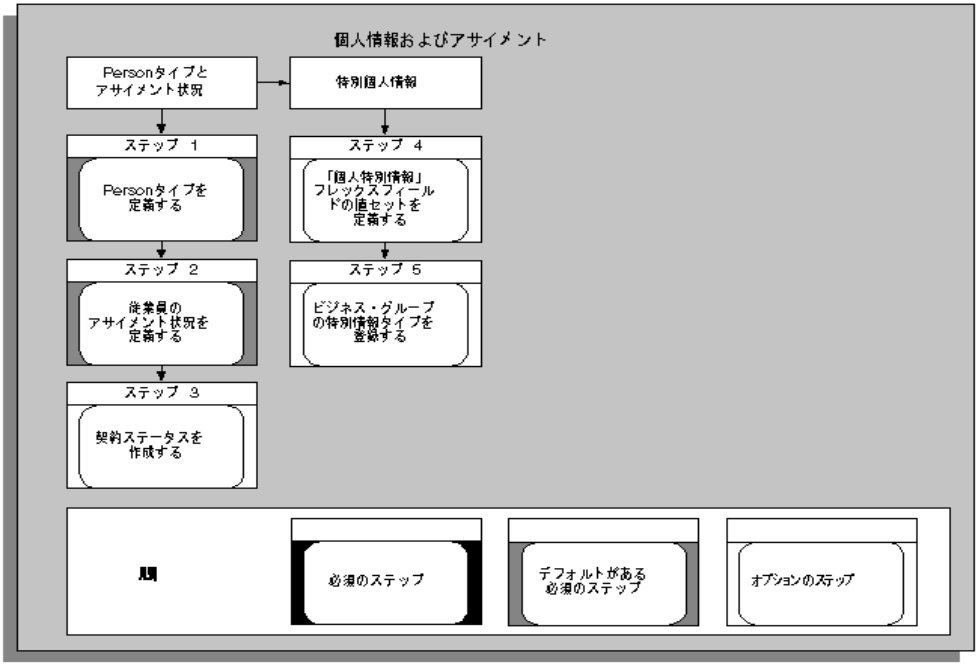
**導入フローチャート : 総合報酬 3**



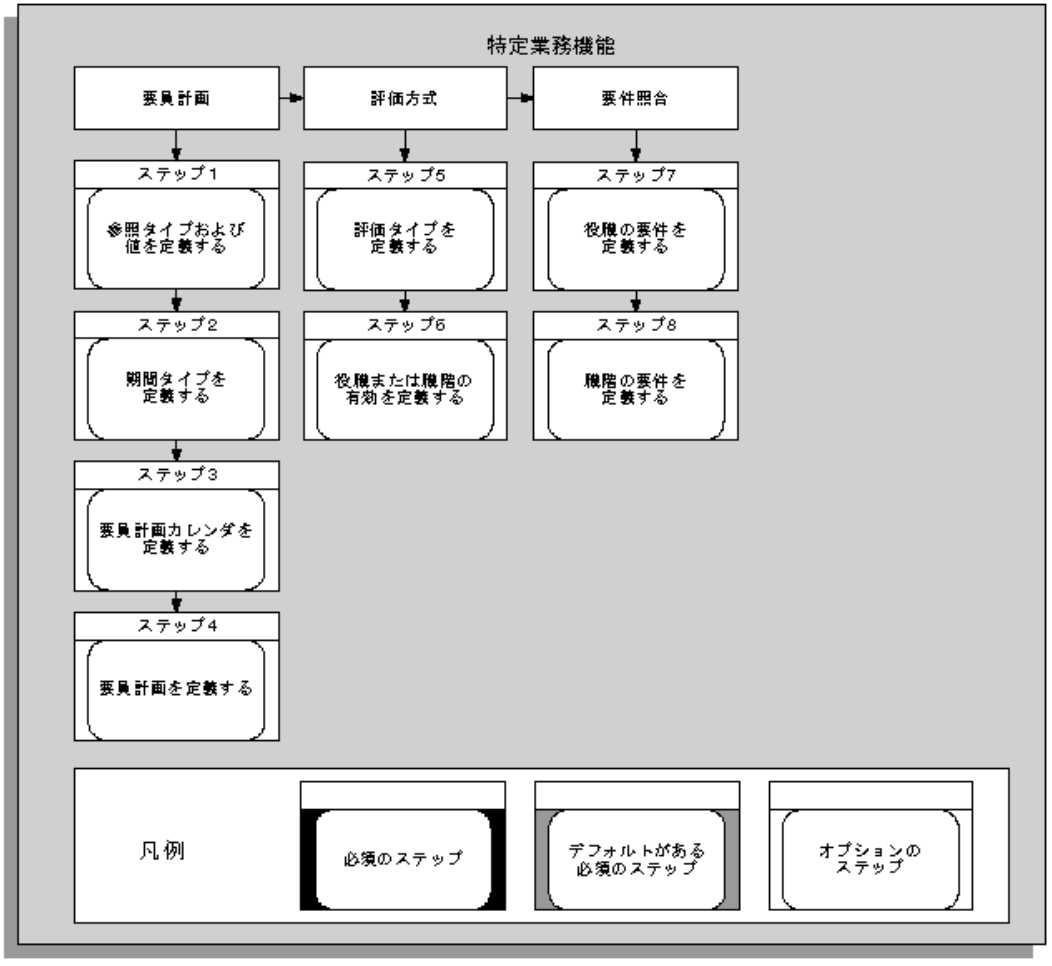
### 導入フローチャート : 総合報酬 4



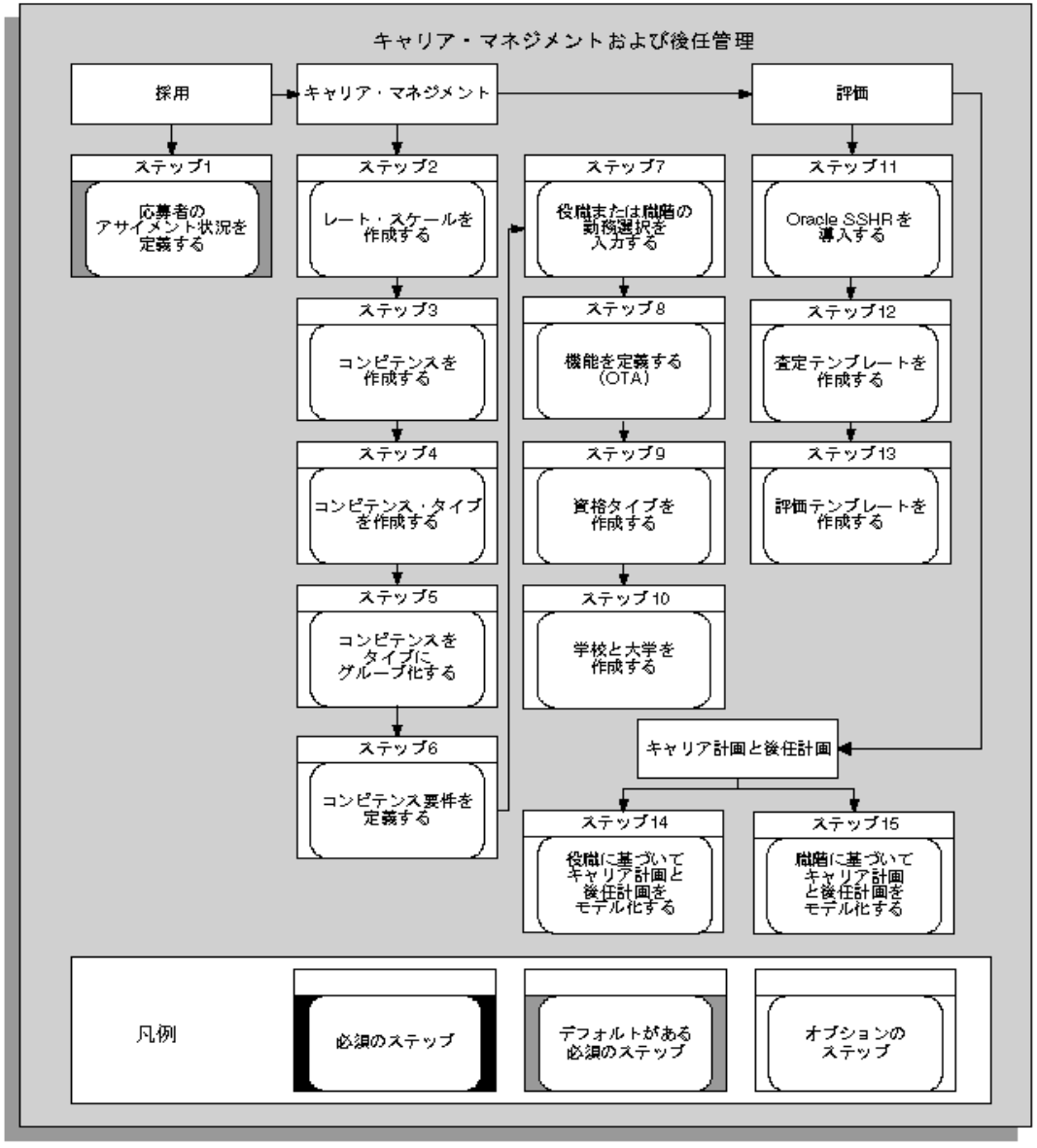

### 導入フローチャート : 個人情報およびアサイメント



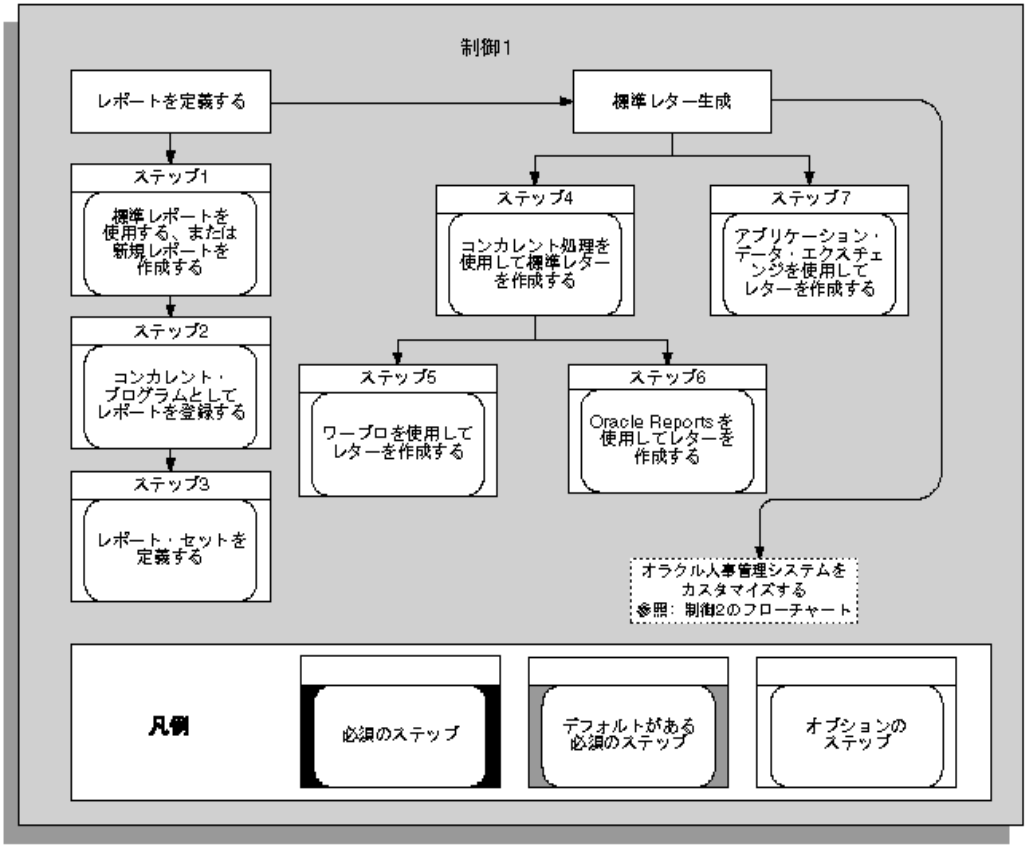
**導入フローチャート : 特定業務機能**



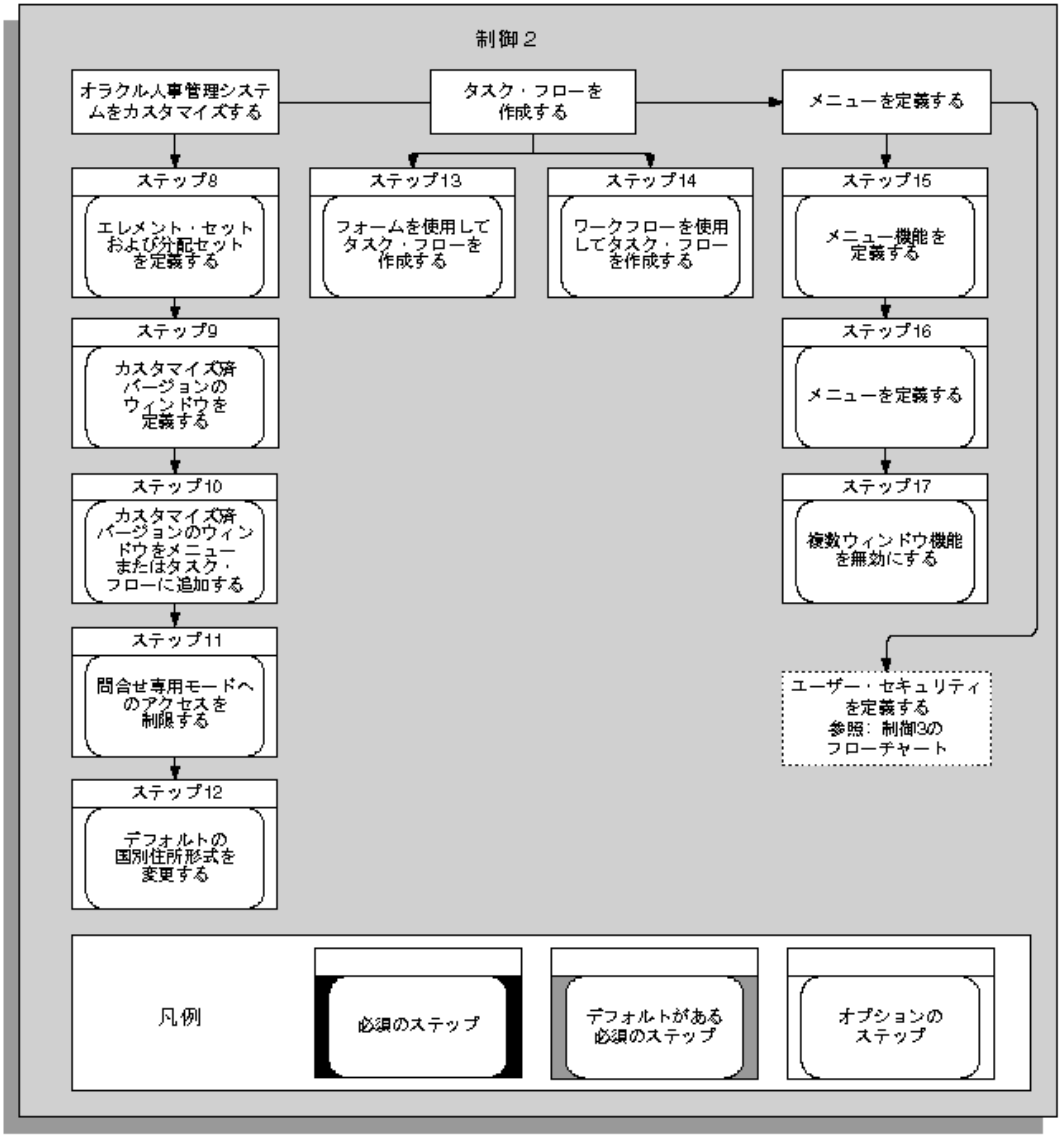
導入フローチャート : キャリア・マネジメントおよび後任管理



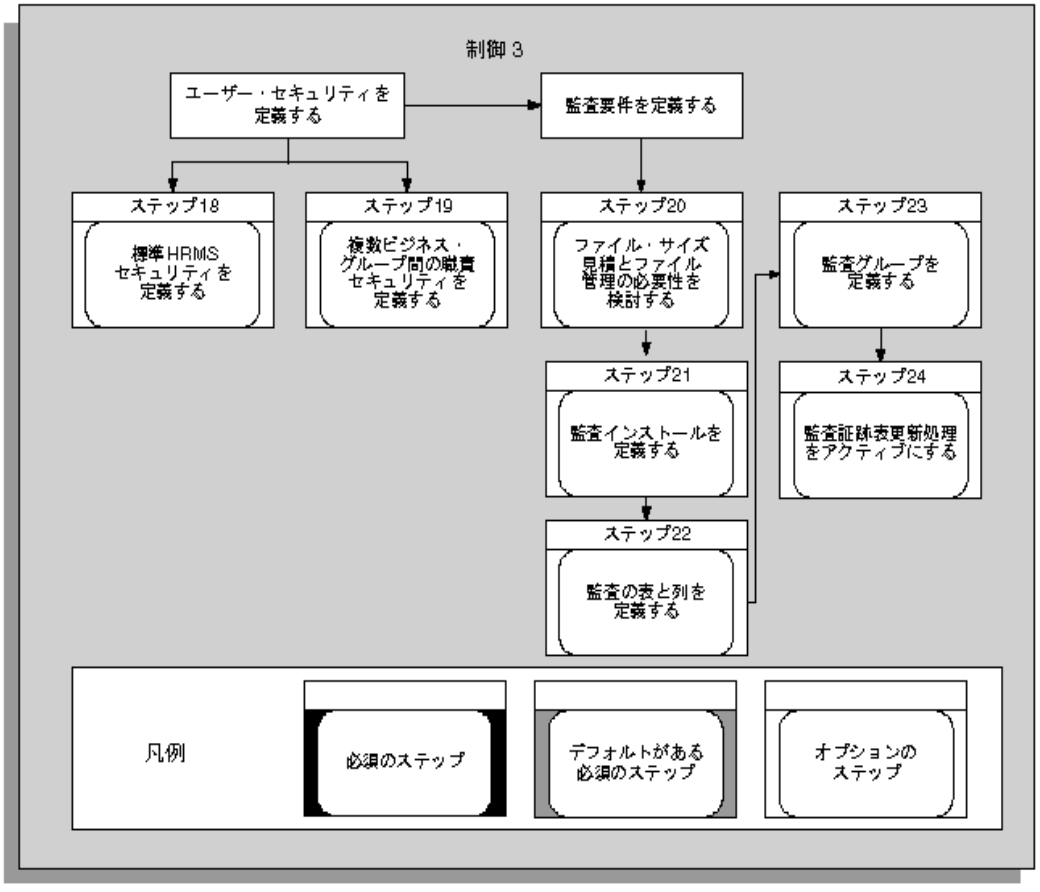
**導入フローチャート : 制御 1**



**導入フローチャート : 制御 2**



**導入フローチャート : 制御 3**



# インストール後のステップ

オラクル人事管理システム リリース 11i には、3 つのポスト・インストール・ユーティリティがあります。

- DataInstall を使用すると、Oracle Human Resources と Oracle Payroll の両方または Oracle Human Resources のみにインストールするすべての国別仕様を指定できます。したがって、これ以降にインストールまたはアップグレードを実行するときは、1 回の操作で国別仕様をインストールできます。DataInstall には、国別仕様と製品の組合せを指定できる一連のメニューが用意されています。
- AutoPatch（adpatch）は、前に DataInstall で指定したインストールまたはアップグレードの組合せに適用されます。
- Quantum をインストールします。Quantum は、Vertex 社製の税務処理製品で、Oracle Payroll（米国）で使用されています。このステップは、米国仕様の Oracle Payroll をインストールする場合にのみ実行します。

**注意：**アップグレードを実行する場合、最初の 2 つのステップは、アップグレード・プロセスの中で実行されます。

**注意：**日本仕様のオラクル人事務管理システムをインストールするためには、前提条件として NLS-Japanese Release のインストールを完了しておく必要があります。

## DataInstall

### ステップ 1 DataInstall ユーティリティを実行する（必須）

DataInstall を使用して、次の手順で国別仕様を指定します。

1. Java ユーティリティ DataInstall を実行して、国別仕様を指定します。次のコマンドを実行してください。

```
java oracle.apps.per.DataInstall <APPS Username> <APPS password>
```

**注意：**複数の会計帳簿をインストールする場合は、最初の APPS アカウントのユーザー名とパスワードを指定してください。

DataInstall のメイン・メニューが表示されます。

2. オプション 1 を選択します。選択できる製品ローカライズの組合せが表示されます。

   データベースに国別仕様データのある製品またはローカライズの場合は、デフォルトの処理がアップグレードになります。これは変更できません。

3. 導入する新しいインストールを選択します。たとえば、3 番のカナダの給与システムをインストールする場合は、3I と入力します。これにより、依存関係が維持された状態で、処理が「Canada Human Resources to Install」に設定されます。

   国別仕様を追加でインストールする場合は、「消去」オプションを使用して誤りを訂正します。追加の Payroll および HR 国別仕様のインストールを選択した場合は、Payroll 国別仕様を消去すると、HR 国別仕様も消去されます。

   アップグレードの場合は「Force Install」を使用できません。「Force Install」は、適用済みのグローバル国別仕様ドライバのステップを再度適用する場合のみ実行する必要があります。

4. 米国または英国以外のローカライズを選択すると、メイン・メニューに戻ります。

   米国または英国のローカライズを選択すると、「DataInstall - College Data Option」画面が表示され、米国および英国ローカライズに大学データが正しくインストールされているかどうかが示されます。インストール・オプションは、既存の大学データがない場合のみ選択できます。データが存在する場合は、ローカライズのデフォルトはアップグレードになりますが、このデフォルトは変更できます。

   既存のデータを残してアップグレードを適用しない場合は「Remain」を選択します。そうでない場合は、「消去」を選択して処理を NULL に設定します。

   ここで「Force Install」は使用できません。

   「戻る」を押して、メイン・メニューに戻ります。ここでオプション 1 を選択すると、さらに変更を加えることができます。オプション 2 を選択すると終了します。

5. 終了を選択すると、「DataInstall Actions Confirmation」画面が表示されます。

   「Y」を選択して変更を保存してから終了するか、「N」を選択して変更を保存せずに終了します。

   終了すると、「DataInstall Actions Summary」画面が表示されます。ここでは、プログラム終了時または ADPATCH がグローバル国別仕様ドライバで実行されたときの処理の要約が表示されます。

# AutoPatch（adpatch）

## ステップ2グローバル国別仕様ドライバを実行する（必須）

汎用 HR ポスト・インストール・ドライバは、汎用のエンティティ範囲と選択したすべてのローカライズを配布します。以下のコマンドを入力して実行します。

```
$ cd $PER_TOP/admin/driver
$ adpatch
```

hrglobal.drv ドライバを適用します。

### グローバル国別仕様ドライバの適用後のステップ

出力ファイル hrlegend.lst を調べます。このファイルには、Java ユーティリティで選択され、このドライバでは適用されていないローカライズが記録されています。インストール・ガイドを参照して、すべて正しく適用されているかどうかを確認するか、オラクル社ワールド・ワイド・サポートまでお問合せください。

### 英国仕様の場合

次の出力ファイルを調べます。

- pegbutcl.lst。このファイルには、最新のバージョンを適用する前に英国国別仕様のシード・ユーザー表を削除するステップが記録されています。また、新しいリリースで変更されたシード・データ名も表示されます。
- perleggb.lst。このファイルには、最新バージョンの適用後に、重複する英国シード・データを取り除くハウスキーピング・ステップが記録されています。また、Oracle Human Resources からオラクル人事管理システムにアップグレードした後に挿入された新しいバランス挿入も記録されています。
- FFXBCP Formula のコンパイル・ステップで生成されるログ・ファイル。FFXBCP ログ・ファイルの名前は、<request_id> log という命名規則に準拠しています。これは、adpatch ログの最後のセクションにあります。

これらのファイルは、オラクル社サポート・サービスがアップグレード後のシードデータでの問題を診断するために使用されます。SQL エラーは重大な問題です。このファイルを参照用として保存し、英国シード・データに問題が起きた場合に使用します。

# Quantum をインストールする

## ステップ 3Oracle Payroll（米国）に Quantum をインストールする（条件付き）

1. Quantum 製品を格納するディレクトリ構造を設定します。

   デフォルトでは、Oracle Payroll は Quantum 製品を $PAY_TOP/vendor/quantum ディレクトリで検索しますが、デフォルトの格納場所を変更することもできます。

   **提案 :** Quantum 製品と Oracle Payroll のリリース・サイクルは異なる可能性があるので、$PAY_TOP/vendor/quantum_versions ディレクトリと、現在のバージョンの Quantum を指す $PAY_TOP/vendor/quantum シンボリック・リンクを作成してください。

2. CD から Quantum コンポーネントを展開します。

   Oracle Applications の CD には、pay というディレクトリに給与計算用の圧縮ファイルがいくつか入っています。Oracle Payroll（米国）がサポートするオペレーティング・システムごとに、1 つの圧縮ファイルがあります。必要なファイルをステップ 1 で作成したディレクトリ構造に展開します。

   ```
   $ cd $PAY_TOP/vendor/quantum_versions
   $ tar xvf sunos.tar
   $ ln -s $PAY_TOP/vendor/quantum_versions/2.2.4 $PAY_TOP/vendor/quantum
   ```

   圧縮ファイルを展開すると、2.2.4 というディレクトリと、2.2.4/lib および 2.2.4/utils というサブディレクトリが作成され、それぞれのディレクトリにファイルが展開されます。2.2.4/devenv というファイルは $FND_TOP/usrxit/devenv ファイルと同じですが、いくつかの行がコメントになっていません。コメントでない行には、Oracle Payroll プロセス PYUGEN をリンクする方法が指定されています。コメントになっていない行は次のとおりです。

   ```
   VND_VERTEX='$(PAY_TOP)/vendor/quantum'
   VND_LINK='$(VND_VERTEX)/lib/libvprt.a ¥
             $(VND_VERTEX)/lib/libqutil.a ¥
             $(VND_VERTEX)/libloc.a ¥
             $(VND_VERTEX)/lib/libcb63.a'
   VNDPAYSL='$(PAY_TOP)/lib/py3c.o $(PAY_TOP)/lib/py3v.o $(VND_LINK)'
   VNDPAYPL='$(PAY_TOP)/lib/py3c.o $(PAY_TOP)/lib/py3v.o $(VND_LINK)'
   export VND_VERTEX VND_LINK VNDPAYPL VNDPAYSL
   ```

   **注意 :** これらの設定のいくつかは、Quantum 製品の格納場所に関する設定です。したがって、Quantum 製品が $PAY_TOP/vendor/quantum にない場合は、このファイルを編集する必要があります。

$FND_TOP/usrxit/devenv ファイルに変更を加えた場合は、これらの変更をファイルに反映させてください。変更を行っていない場合は、2.2.4/devenv を $FND_TOP_usrxit/devenv にコピーしてください。

3. adrelink を使用して、Oracle Payroll の実行可能ファイル PYUGEN を再リンクします。

```
$ adrelink force=y ranlib=y "pay PYUGEN"
```

ログ・ファイルをチェックして、adrelink が正常に完了したことを確認します。

4. Quantum 製品のデータ・ファイルを作成します。

Quantum データ・ファイルを作成するには、最初にデータ・ファイルを格納するディレクトリを作成します。Oracle Payroll では、これらのデータ・ファイルが $PAY_TOP/vendor/quantum/data にあることが想定されています。

次に、Quantum utils ディレクトリにある dbcreate ユーティリティを実行します。このユーティリティを実行すると、Payroll または Geocoder のメニューが表示されます。Payroll オプションを選択すると、"Enter the Payroll datasource name:" というプロンプトが表示されます。データ・ファイルの格納場所を入力します。たとえば、/apps/pay/11.5/vendor/quantum/data のように入力してください。処理が完了すると、メニューが再表示され、ここでユーティリティを終了できます。

---

**注意：**データ・ファイルのファイル権限が、関連するすべてのユーザーに対して読込み可能に設定されていることを確認してください。このように設定されていないと、Oracle Payroll はこれらのファイルにアクセスできません。

---

5. Quantum データ・ファイルに移入します。

作成されたデータ・ファイルに、税務データを移入する必要があります。税務データは CD の qfpt.dat ファイルに入っています。このファイルを CD から Quantum 製品領域にコピーします。移入が完了すると、データ・ファイル更新ユーティリティを実行できます。このユーティリティは、vprtmupd という utils ディレクトリにあります。メニューから「Update Payroll Tax」オプションを選択し、画面に表示された質問に答えます。最初にデータソースが要求されます。これには、前のステップで作成したデータ・ファイルの位置を指定します。次に、qfpt.dat ファイルの位置を指定します。次に例を示します。

```
Enter Datasource:/apps/pay/11.5/vendor/quantum/data
Enter the path of the update file:/apps/pay/11.5/vendor/quantum
```

**注意 :** CD に入っている更新ファイルはデフォルトのファイルですが、このファイルを使用しても税金が正しく計算されるとは限りません。このファイルは、テストを目的としています。適切な更新ファイルは Vertex 社から入手してください。

6. Quantum データ・ファイルの位置を登録します。

   Quantum のデータ・ファイルがデフォルトの位置（$PAY_TOP/vendor/quantum/data）にない場合は、これらのファイルの位置を Oracle Payroll に指定する必要があります。これを指定するには、PAY_ACTION_PARAMETERS 表に次の行を追加します。

```
SQL> insert into pay_action_parameters
2 values ('TAX_DATA', '/apps/quantum/data');
```

# マスター・データのロード

## ステップ 4マスター・データをロードする（日本仕様の場合のみ）

必要に応じて、住所、銀行、学校のマスター・データをロードします。

1. 住所マスター・データのロード

   住所データは SQL*LOADER 等を使用して PER_JP_ADDRESS_LOOKUPS 表にロードします。住所データは既存システムのデータをそのままご使用になるか、または郵政省のホームページから最新の住所データがダウンロード可能です。

2. 銀行マスター・データのロード

   銀行マスター・データは銀行データは PER_JP_BANKS 表に、支店データは PER_JP_BRANCES 表に SQL*LOADER 等を使用してロードしてください。銀行マスター・データは既存システムのデータをそのままお使いください。

3. 学校マスター・データのロード

   学校マスター・データは PERJ_JP_SCHOOL_LOOKUPS 表に SQL*LOADER 等を使用してロードしてください。学校マスター・データは、既存システムのデータをそのままご使用ください。

# 2

# 導入ステップ

オラクル人事管理システムでは、Oracle Human Resources、Oracle Payroll、Oracle Advanced Benefits、Oracle Self-Service Human Resources（SSHR）のいずれも、業務ニーズに応じた柔軟な導入プロジェクト計画の策定が可能です。

オラクル人事管理システムをはじめに導入する機能を選択します。その他の機能は必要に応じて導入します。

たとえば、まず人事システム・ユーザー向けに導入し、次の段階で給与関連を追加します。もしくは、最初に給与計算機能を導入することも可能です。人事システムの情報や機能の範囲を後に拡張することもできます。

どの導入プロセスにおいても、方針を決定しておくことが重要です。オラクル人事管理システムをカスタマイズする前に、システムの使用方法を検討する必要があります。

段階的、部分的な導入方法を採用することで、システムを効率的に使用できます。

オラクルでは、コンサルタントがユーザー・ニーズに応じた研修や技術的、専門的知識を提供しています。業務ニーズに応じた人事管理システムを迅速かつ効率的に導入できます。

## 始める前に

オラクル人事管理システムを導入する前に、国別仕様の起動データがインストールされていることを確認してください。通常、このインストールは、システム管理者が実行します。この起動データは、エレメント、支払方法、国別仕様のフレックスフィールド体系を使用する前に必要になります。

詳細は、『Oracle Applications インストレーション・マニュアル』を参照してください。

また、オラクル人事管理システムの実行を開始する前に実行する必要のある導入後のステップがあるかどうかも確認してください。

**関連項目：** 1-1 ページ「導入プラン」

## Oracle Applications Implementation Wizard

複数の Oracle Applications 製品を実行する場合は、Oracle Application Implementation Wizard を使用して設定アクティビティを調整することをお薦めします。Implementation Wizard を使用すると、導入したアプリケーションを設定するステップが示され、製品間での導入の依存関係を満たし、重複する設定ステップが少なくなるような論理的順序が提示されます。

Implementation Wizard を使用して、設定ステップのグラフィカルな概要の表示、設定アクティビティのオンライン・ヘルプの参照および適切な設定ウィンドウのオープンを行うことができます。Wizard を使用して各ステップのコメントを記録することにより、導入を文書化して参照、レビューできます。

関連資料 :『Oracle Applications Implementation Wizard ユーザーズ・ガイド』

# 管理

管理ステップは通常、システム管理者が実行します。システム管理者のユーザー名とパスワードを使用してシステムにログインします。ユーザー名やパスワードが不明な場合は、データベース管理者に連絡してください。

## キー・フレックスフィールドを定義する

オラクル人事管理システムには、次に示すように、ビジネス・グループを定義する前に設定が必要なキー・フレックスフィールドが 5 つあります。

- 役職
- 職階
- 等級
- People グループ
- 原価割当

このような 5 つのキー・フレックスフィールドを導入する前に、必須条件を明確に指定しておく必要があります。この指定内容には、各キー・フレックスフィールドに対して次の詳細を含める必要があります。

- 体系名とセグメント数
- フレックスフィールドのセグメントの名称、順序、検証オプション、識別子
- 使用するフレックスフィールドの値セットと値リスト

キー・フレックスフィールドの定義を完了した後で、キー・フレックスフィールド・データベース・アイテムの作成コンカレント・プロセスを実行して、フレックスフィールドの個々のセグメントのデータベース・アイテムを生成する必要があります。

このことは、役職、職階、等級および People グループのキー・フレックスフィールドにのみ適用されます。

## 「役職」フレックスフィールドを定義する

企業の役職情報の入力、レポート作成に、オラクル人事管理システムの柔軟性を最大限に活用するには、必要な条件を特定してから、次の順序で導入を開始します。

## ステップ 1「役職」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 2「役職」フレックスフィールドのセグメントを定義する

ビジネス・グループに使用するセグメントを保持する、「役職」フレックスフィールド体系を定義します。この体系は「役職」ウィンドウで一意の役職名を作成するために使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。このチェック・ボックスをブランクにすると、「役職」ウィンドウで新しい役職名の組合せを作成できません。

> **注意：**セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、最大 150 文字までの英数字を自由に入力できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

### ステップ 3「役職」フレックスフィールドのセグメント値を定義する

「役職」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

### ステップ 4「役職」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲が定義できます。

「相互検証ルール」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

### ステップ 5「役職」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せに、デフォルト・オプションを指定する場合は、このような組合せに別名を定義します。

「短縮別名」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

### ステップ 6「役職」フレックスフィールド体系を確定、コンパイルする

これで「役職」フレックスフィールド定義を確定する準備が整いました。「キー・フレックスフィールド・セグメント」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドに「Yes」を入力して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「役職」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールを「役職フレックスフィールド」ウィンドウで使用できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 7キー・フレックスフィールド・データベース・アイテムの作成プロセスを実行する

フレックスフィールドの個々のセグメントを個別のデータベース・アイテムとして使用するときは、「新規要求の発行」ウィンドウからコンカレント・プロセスを実行します。このプロセスに必要なパラメータはキー・フレックスフィールド名だけです。

「新規要求の発行」ウィンドウを使用してください。

**関連項目：**『Oracle Applications ユーザーズ・ガイド』の「要求の発行」

## 「職階」フレックスフィールドを定義する

企業の職階情報の入力、レポート作成にオラクル人事管理システムの柔軟性を最大限に活用する必須条件を指定してから、次の順序で実行します。

## ステップ 8「職階」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 9「職階」フレックスフィールドのセグメントを定義する

ビジネス・グループに使用するセグメントを保持する、「職階」フレックスフィールド体系を定義します。この体系は「職階」ウィンドウで一意の職階名を作成するために使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。このチェック・ボックスをブランクにすると、「職階」ウィンドウで新しい職階名の組合せを作成できません。

**注意：**セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、最大 150 文字までの英数字を自由に入力できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 10「職階」フレックスフィールドのセグメント値を定義する

「職階」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値の定義」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 11「職階」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲が定義できます。

「相互検証ルール」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

## ステップ 12「職階」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せに、デフォルト・オプションを指定する場合は、このような組合せに別名を定義します。

「短縮別名」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

## ステップ 13「職階」フレックスフィールド体系を確定、コンパイルする

これで「職階」フレックスフィールド定義を確定する準備が整いました。「キー・フレックスフィールド・セグメント」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドに「Yes」を入力して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「職階」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールを「職階フレックスフィールド」ウィンドウで使用できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 14 キー・フレックスフィールド・データベース・アイテムの作成プロセスを実行する

フレックスフィールドの個々のセグメントを個別のデータベース・アイテムとして使用するときは、「新規要求の発行」ウィンドウからコンカレント・プロセスを実行します。このプロセスに必要なパラメータはキー・フレックスフィールド名だけです。

「新規要求の発行」ウィンドウを使用してください。

**関連項目：**『Oracle Applications ユーザーズ・ガイド』の「要求の発行」

## 「等級」フレックスフィールドを定義する

企業の等級情報の入力、レポート作成にオラクル人事管理システムの柔軟性を最大限に活用する必須条件を指定してから、次の順序で実行します。

## ステップ 15「等級」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 16「等級」フレックスフィールドのセグメントを定義する

ビジネス・グループに使用するセグメントを保持する、「等級」フレックスフィールド体系を定義します。この体系は「等級」ウィンドウで一意の等級名を作成するために使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。このチェック・ボックスをブランクにすると、「等級」ウィンドウで新しい等級名の組合せを作成できません。

**注意：**セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、最大 150 文字までの英数字を自由に入力できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 17「等級」フレックスフィールドのセグメント値を定義する

「等級」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値の定義」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 18「等級」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲が定義できます。

「相互検証ルール」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

## ステップ 19「等級」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せに、デフォルト・オプションを指定する場合は、このような組合せに別名を定義します。

「短縮別名」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

## ステップ 20「等級」フレックスフィールド体系を確定、コンパイルする

これで「等級」フレックスフィールド定義を確定する準備が整いました。「キー・フレックスフィールド・セグメント」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドに「Yes」を入力して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「等級」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールを「等級フレックスフィールド」ウィンドウで使用できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 21 キー・フレックスフィールド・データベース・アイテムの作成プロセスを実行する

フレックスフィールドの個々のセグメントを個別のデータベース・アイテムとして使用するときは、「新規要求の発行」ウィンドウからコンカレント・プロセスを実行します。このプロセスに必要なパラメータはキー・フレックスフィールド名だけです。

「新規要求の発行」ウィンドウを使用してください。

**関連項目：**『Oracle Applications ユーザーズ・ガイド』の「要求の発行」

## 「People グループ」フレックスフィールドを定義する

People グループ情報は従業員アサイメントに関連付けられているため、組合員など、特別な従業員グループの識別に使用します。

**警告：オラクル人事管理システムの「People グループ」キー・フレックスフィールドに少なくとも 1 つのセグメントを定義する必要があります。**

セグメントを 1 つも定義しないと、従業員または応募者に対して「アサイメント」ウィンドウを使用できません。

企業の People グループ情報の入力、レポート作成にオラクル人事管理システムの柔軟性を最大限に活用する必須条件を指定してから、次の順序で実行します。

## ステップ 22「People グループ」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 23「People グループ」フレックスフィールドのセグメントを定義する

ビジネス・グループに使用するセグメントを保持する、「People グループ」フレックスフィールド体系を定義します。この体系は「アサイメント」ウィンドウで People グループの詳細を入力するために使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。このチェック・ボックスをブランクにすると「アサイメント」ウィンドウで People グループ情報を入力できません。

> **注意 :** セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、最大 150 文字までの英数字を自由に入力できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 24「People グループ」フレックスフィールドのセグメント値を定義する

「People グループ」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値の定義」ウィンドウを使用してください。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 25「People グループ」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲が定義できます。

「相互検証ルール」ウィンドウを使用してください。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

## ステップ 26「People グループ」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せに、デフォルト・オプションを指定する場合は、このような組合せに別名を定義します。

「短縮別名」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

## ステップ 27「People グループ」フレックスフィールド体系を確定、コンパイルする

これで「People グループ」フレックスフィールド定義を確定する準備が整いました。「キー・フレックスフィールド・セグメント」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドに「Yes」を入力して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「People グループ」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールを「People グループ・フレックスフィールド」ウィンドウで使用できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 28 キー・フレックスフィールド・データベース・アイテムの作成プロセスを実行する

フレックスフィールドの個々のセグメントを個別のデータベース・アイテムとして使用するときは、「新規要求の発行」ウィンドウからコンカレント・プロセスを実行します。このプロセスに必要なパラメータはキー・フレックスフィールド名だけです。

「新規要求の発行」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「キー・フレックスフィールド・データベース・アイテムの作成」

## 「原価割当」フレックスフィールドを定義する

原価割当情報は、給与計算結果を基に従業員負担計算の詳細を入力する目的に通常使用します。Oracle Payroll をインストールしている場合は、給与計算結果を基に原価を累計して、Oracle General Ledger に転送できます。Oracle Payroll をインストールしていない場合は、原価割当情報の入力に原価計算フレックスフィールドを使用できます。

企業の原価計算情報の入力、レポート作成にシステムの柔軟性を最大限に活用する必須条件を指定してから、次の順序で実行します。

**警告：オラクル人事管理システムでは、「原価割当」キー・フレックスフィールドに少なくとも 1 つのセグメントを定義する必要があります。セグメントを 1 つも定義しないで、フレックスフィールド・ウィンドウでウィンドウを使用すると、エラーの原因になります。**

## ステップ 29「原価割当」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 30「原価割当」フレックスフィールドのセグメントと識別子を定義する

ビジネス・グループに使用するセグメントを保持する「原価割当」フレックスフィールド体系を定義します。この体系はオラクル人事管理システムに給与原価計算詳細を入力するために使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。このチェック・ボックスをブランクにすると、システムに原価計算詳細を入力できません。

**注意：**セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、最大 150 文字までの英数字を自由に入力できます。

オラクル人事管理システムのキー・フレックスフィールドには、識別子が必要な「原価割当」フレックスフィールドがあります。セグメント識別子を使用して、システムに入力可能な原価計算情報のレベルを制御できます。識別子によって、入力可能な原価計算情報のレベルが決まります。各セグメントには次の 6 つの識別子があります。

| 識別子 | ウィンドウ上での効果 |
|---|---|
| Payroll | 「給与」ウィンドウでセグメント値を入力する。 |
| Link | 「エレメント・リンク」ウィンドウでセグメント値を入力する。 |
| Balancing | 「エレメント・リンク」ウィンドウでバランス・セグメント値を入力する。 |
| Organization | 組織の「原価計算情報」ウィンドウでセグメント値を入力する。 |
| Assignment | 組織の「原価計算」ウィンドウでセグメント値を入力する。 |
| Entry | 「エレメント・エントリ」ウィンドウでセグメント値を入力する。 |

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 31「原価割当」フレックスフィールドのセグメント値を定義する

「原価割当」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値の定義」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 32「原価割当」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを、追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲が定義できます。

「相互検証ルール」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

### ステップ 33「原価割当」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せに、デフォルト・オプションを指定する場合は、このような組合せに別名を定義します。

「短縮別名」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

### ステップ 34「原価割当」フレックスフィールド体系を確定、コンパイルする

これで「原価割当」フレックスフィールド定義を確定する準備が整いました。「キー・フレックスフィールド・セグメント」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドに「Yes」を入力して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「原価割当」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールを「原価割当」フレックスフィールド」ウィンドウで使用できます。

「キー・フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## 付加フレックスフィールドを定義する

標準ウィンドウに独自のフィールドを追加するには、オラクル人事管理システムの付加フレックスフィールドを使用して定義します。たとえば、ある従業員の運転免許証番号を入力する場合、この追加情報を入力する追加個人詳細フレックスフィールドのセグメントを定義します。

セグメントの定義をすると、「個人情報」ウィンドウに標準の個人情報と運転免許証番号を入力できます。

**警告：付加フレックスフィールドは実表レベルで定義されます。そのため、実表を使用するどのウィンドウにも同じ付加フレックスフィールドのセグメントが表示されます。この例では、運転免許証番号は「個人情報」ウィンドウと「連絡先」ウィンドウの両方に表示されます。**

付加フレックスフィールドを実行する前に、必須条件を明確に指定する必要があります。必須条件には、次の詳細を含める必要があります。

- コンテキストと各コンテキストのセグメント数
- フレックスフィールドのセグメントの名称、順序、検証オプション
- 使用するフレックスフィールドの値セットと値リスト

定義する付加フレックスフィールドには次の 2 つのタイプがあります。

- **グローバル・セグメント**

  フレックスフィールド・ウィンドウに常に表示されるセグメント

- **コンテキスト依存セグメント**

  定義されているコンテキストが入力されるとセグメントが表示されます。ユーザーにコンテキストの指定を促すプロンプトを表示したり、同じブロックの参照フィールドから自動的にコンテキストを挿入できます。

**提案 :** 多くの場合、コード、「実表」フィールド、内容や摘要の入ったフィールドから選択できます。コンテキスト依存セグメントの参照フィールドには、常に「実表」フィールドを使用してください。このようなフィールドは通常、実表の列と同じ名称です。

システムに組み込まれている標準レポートには、付加セグメント値が含まれています。前述した提案に従って設定した場合、定義したプロンプトがレポートに表示使用されます。そうでない場合は、システム定義のプロンプトが表示されます。

**提案 :** デートトラック履歴の参照リストに、付加フレックスフィールドのセグメント値を含めるには、システム定義のデートトラック履歴表示を変更する必要があります。

## ステップ 35参照フィールドを登録する

アプリケーション開発者の職責を使用して、付加フレックスフィールドの定義を更新する必要があります。「付加フレックスフィールド」ウィンドウから、参照フィールド・ブロックに移動して、使用する参照フィールドの名称を入力します。

**警告 : 付加フレックスフィールドには、あらかじめ定義されている変更禁止のものがあります。このような付加フレックスフィールドは、国別仕様情報や住所形式に対処しています。**

変更禁止のフレックスフィールドの定義を変更しないでください。この定義は、オラクル人事管理システムの基本的な部分ですので、変更するとシステムの動作異常が発生する恐れがあります。

将来的には、オラクル人事管理システムでこれらのフレックスフィールドの他のセグメントを使用する可能性もあります。変更禁止のフレックスフィールドには、絶対にセグメントを

追加しないでください。このような操作をすると、将来システムをアップグレードするときに影響が出る恐れがあります。

「付加フレックスフィールド」ウィンドウを使用してください。

## ステップ 36フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

- 値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。
- 値セットの属性は、値の検証方法も制御します。

---

**注意 :** 値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

---

「値セット」ウィンドウを使用してください。

**関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 37付加フレックスフィールドのセグメントを定義する

各コンテキストの付加フレックスフィールドのセグメントを定義します。

セグメントの検証に値セットを使用する必要はありません。値セットを指定しなければ、ユーザーは 150 文字までの英数字を自由に入力できます。

1. フレックスフィールド・ウィンドウに常に表示されるセグメントを定義するには、グローバル・コンテキストを使用する。
2. 特定のコンテキストにしか表示されないセグメントを定義するには、そのコンテキスト名を入力する。
3. 付加フレックスフィールド定義を確定、コンパイルする。

---

**警告 : あるセグメントを「必須」として定義すると、このセグメントはシステムの各レコードに対して必須になります。よく発生する問題として次の 2 つがあります。**

---

- レコード入力後に「必須」セグメントを定義するとき。既存のレコードはこのセグメントに値がないため、既存のレコードを問い合せるとエラー・メッセージが表示されます。
- いくつかの付加フレックスフィールドは、複数のブロックで使用されています。たとえば、追加個人詳細の「必須」セグメントは各従業員、応募者、連絡先に対してすべて入力する必要があります。

「付加フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「付加フレックスフィールド体系の定義」

## ステップ 38フレックスフィールドのセグメント値を定義する

付加フレックスフィールドが使用する値セットに「独立」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値の定義」ウィンドウを使用してください。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 39付加フレックスフィールド・データベース・アイテムの作成プロセスを実行する

付加フレックスフィールドを定義した場合、非コンテキスト依存の付加フレックスフィールドのセグメントのデータベース・アイテムを作成するために、「付加フレックスフィールド・データベース・アイテム」を実行する必要があります。

非コンテキスト依存の付加フレックスフィールドのセグメントを追加作成した場合、「付加フレックスフィールド・データベース・アイテム作成」処理を再実行する必要があります。

---

**注意：**コンテキスト依存のフレックスフィールドのセグメントのデータベース・アイテムが必要なときは、詳しいデータベース・アイテムの追加方法をオラクル社のサポート担当者にお問い合せください。

---

「新規要求の発行」ウィンドウを使用してください。

**関連項目：**『Oracle Applications ユーザーズ・ガイド』の「要求の発行」

# その他情報タイプ（EIT）を定義する

その他情報タイプは、オラクル人事管理システムの最も重要な 6 つのエンティティに、情報タイプを数に制限なく追加するための付加フレックスフィールド・タイプです。

たとえば、アサイメントの EIT を使用して、アサイメントのプロジェクト業務に関する情報を保持できます。

**注意：**組織では、分類によって EIT をグループ化し、ユーザーが分類を選択したときに、その分類に関連する EIT が表示されるようにできます。このため、組織で EIT を導入するには、追加ステップがいくつか必要です。

## その他情報タイプを定義する（組織以外）

### ステップ 40事業所、役職、職階、個人情報およびアサイメントの「その他情報タイプ」を定義する

必要なその他情報タイプを決定した後で、付加フレックスフィールドをタイトルで選択する必要があります。「コンテキスト・フィールド値」リージョンに新規レコードを作成し、「コード」フィールドに新規情報タイプの名称を入力します。セグメント値を入力し、付加フレックスフィールドをコンパイルします。

「付加フレックスフィールド・セグメント」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「その他情報タイプの設定（組織の EIT 以外）」

### ステップ 41その他情報タイプの職責アクセスを設定する

EIT は、どの職責にも自動的には表示されません。EIT の職責レベル・アクセスを設定する必要があります。あるいは、カスタムフォーム・セキュリティを使用して、個々の EIT を特定のタスクフロー・ウィンドウに追加します。通常このセキュリティ・レベルは、ユーザーのアクセスを制限する必要があるときに、後で実行の際に定義されます。

**注意：**このセキュリティは、組織の EIT には適用されません。

「情報タイプ・セキュリティ」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「職責に対するその他情報タイプの設定」

### 組織のその他情報タイプを定義する

組織分類の EIT は、他の EIT とは異なる方法で設定されます。組織の EIT を定義するときは、EIT を組織の分類に関連付ける必要もあります。ユーザーが分類を選択すると、システムは適切な EIT セットを表示します。

### ステップ 42組織分類を定義する

この方法で EIT をグループ化するには、新規組織分類を定義します。既存の分類を使用する場合は、新規分類を定義する必要はありません。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

### ステップ 43ステップ 44組織分類のその他情報タイプを設定する

新規 EIT を定義し、HR_ORG_INFORMATION TYPES 表に行を入力します。次に、この EIT を使用可能にする組織分類を指定します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「組織分類のその他情報タイプの設定」

## システム管理

次の作業はシステム管理者が実行します。

### ステップ 45通貨を有効にする

主要通貨はすべて Oracle Applications に定義されています。使用されている通貨コードは、ISO 規格に準拠しています。しかし、基本通貨または報酬と福利厚生情報に対して使用する特定の通貨を有効にする必要があります。

「基本通貨」はビジネス・グループで使用されるデフォルトの通貨です。

**注意：**オラクル人事管理システムでは拡張精度を使用していません。精度は Formula を使用して制御できます。

「通貨」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「通貨の有効化」

### ステップ 46「全表示」人事管理システム・ユーザー職責を定義する

人事管理システム・ウィンドウにアクセスするには、システムで提供されているデフォルトの職責の 1 つにアクセスするアプリケーション・ユーザーを新しく作成する必要があります。

---

**注意 :** 新規ユーザー名とパスワードを定義しますが、デフォルトの職責は選択しないでください。かわりに「セキュリティ・プロファイルの割当」ウィンドウを使用します。

---

「ユーザー」ウィンドウを使用してください。

**関連項目 :** 『Oracle Applications システム管理者ガイド』の「「ユーザー」ウィンドウ」

## アプリケーション・データ・エクスチェンジ（ADE）と階層ダイアグラマ

### ステップ 47 ADE を設定する

オラクル人事管理システムのデータベースと他のアプリケーションの間で情報をエクスポートするアプリケーション・データ・エクスチェンジ（ADE）を設定できます。

**関連項目 :** 『アプリケーション・データ・エクスチェンジおよび階層ダイアグラマ・ユーザーズ・ガイド』の「設定ステップの概要」

### ステップ 48階層ダイアグラマへのアクセスを制御する

グラフィックを利用して、組織や職階の階層を作成し、保守することもできます。これを階層ダイアグラマと呼びます。組織と職階の階層は、会社のレポート・ラインを反映します。階層ダイアグラマは、オラクル人事管理システムの ADE から起動します。階層ダイアグラマを使用するには、まず ADE をインストールする必要があります。

**関連項目 :** 『アプリケーション・データ・エクスチェンジおよび階層ダイアグラマ・ユーザーズ・ガイド』の「階層ダイアグラマの設定」

# ワーク・ストラクチャ

## 組織体系を定義する

### ステップ 49ビジネス・グループを適合または作成する

ビジネス・グループとは組織の特別な区分を表します。各ビジネス・グループは、独自の内部組織や等級、役職、職階、給与、従業員、応募者、報酬と福利厚生に合せて独自のデフォルト値を設定できます。

Setup Business Group は、オラクル人事管理システムに組み込まれています。このビジネス・グループは、デフォルトの職責で使用されます。このビジネス・グループとすべてのデフォルト定義を初期データに使用して独自のビジネス・グループを作成したり、必要に応じて他のビジネス・グループを定義したりできます。

**警告 : Setup Business Group には、US というデフォルト国別仕様コードと、USD というデフォルトの基本通貨があります。**

ビジネス・グループで給与計算を実行する場合、または別の地域の法律を適用する場合は、有効な国別仕様コードと基本通貨を選択して新規ビジネス・グループを作成する必要があります。システムは、これらの値を使用して、その国の法定要件と処理要件に準拠する必要のある事前定義データにコピーします。

このような定義は、一度保存すると変更できません。

「組織」ウィンドウを使用してください。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「ビジネス・グループの適合および新規作成」

### ステップ 50ビジネス・グループの全表示職責を作成する

オラクル人事管理システムに添付の Setup Business Group を使用している場合、このステップはスキップしてかまいません。

「職責」ウィンドウを使用してください。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「すべてのレコードを表示できる職責の定義」

## ステップ 51職責の個別プロファイル・オプション値を設定する

新しい職責用に HR 個別プロファイル・オプションを設定します。「HR: ユーザー・タイプ」オプションを設定する必要があります。

その他の個別プロファイル・オプションも設定できます。

「システム・プロファイル値」ウィンドウを使用してください。

> **関連項目：**『Oracle Applications システム管理者ガイド』の「「システム・プロファイル値」ウィンドウ」

## ステップ 52参照タイプおよび値を定義する

参照はオラクル人事管理システムで多くの値リストを提供します。たとえば、「個人情報」ウィンドウのタイトルと国籍では参照を使用します。

オラクル人事管理システムには、いくつかの参照タイプがあらかじめ定義されています。このようなシステム定義の参照は、値を定義するだけで使用できます。

参照値は、値リストに表示される有効なエントリです。これを利用することで、情報の選択が簡単になり、ユーザーは有効なデータだけをオラクル人事管理システムに入力できます。

随時新しい参照値を追加できます。値の有効フラグを「No」に設定すると、その値は値リストに表示されなくなります。開始日と最終日を使用して、値をリストに表示する時期を制御できます。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 53事業所を作成する

企業で使用している事業所を作成します。各事業所と住所を定義するのは一度だけです。このため、同じ住所の組織がいくつかある場合に、データの入力時間を節約できます。

「事業所」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「事業所の設定」

## ステップ 54組織を作成する

組織とは、すべての企業に共通する基本ワーク・ストラクチャを表します。通常、組織はビジネス・グループ内の業務、管理およびレポート・グループを表示します。

企業の内部組織に加えて、納税事務所や政府のレポート用、第三者への支払用に他の組織も定義できます。

**提案 :** オラクル人事管理システムには、システム定義の組織分類リストが用意されています。この値は、参照タイプ ORG_CLASS 用に定義されているため、「組織」ウィンドウにオプションが表示されます。

実行に不要な参照値は、「アプリケーション・ユーティリティ参照」ウィンドウで無効にできます。

オラクル人事管理システムに、過去のアサイメント情報をロードする場合は、有効日を入力したかどうか確認してください。組織の開始日より前に、従業員を組織に対して割り当てることはできません。

**提案 :** 初期設定では、1951 年 1 月 1 日のような固定した日付を検討してください。こうすることでデータ入力が簡素化されます。

## ステップ 55組織分類と追加情報を入力する

各組識の適切な分類と、その他情報タイプの詳細を入力します。

「組織」ウィンドウを使用してください。

**関連項目 :** ■

- 『Oracle HRMS ユーザーズ・ガイド』の「組織分類の入力」
- 『Oracle HRMS ユーザーズ・ガイド』の「追加情報の入力」

## Oracle Cash Management 統合の会計参照情報

支払調停のために Oracle Payroll を Oracle Cash Management とともに使用している場合は、会計参照情報も設定する必要があります。

組織の営業単位分類を選択し、さらに組織の GRE/ 法的エンティティ分類を選択します。会計帳簿と VAT 登録番号を、法的エンティティ会計のその他情報に入力してください。

「組織」ウィンドウを使用してください。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「組織の作成」

## ステップ 56組織階層を定義する

ビジネス・グループに含める組織の数に制限はありません。管理などのレポート体系は、このような組織を報告階層に割り当てることで表示できます。また、組織は複数の階層に組み込むことができますが、1 つの階層に同じ組織を複数設定することはできません。

**提案 :** 上位の組織にその下位の組織を 1 つ組み合せて、主要な報告階層を定義します。定義後、「組織」ウィンドウで階層への組織の追加を簡単に行えます。

組織のレポート・ラインは頻繁に変更されますが、必要に応じて、開始日と最終日を使用して新しいバージョンの階層を作成できます。こうすることで、組織変更の履歴を保持できるだけでなく、次期の変更計画にもこの機能を利用できます。

デートトラックを使用すると、有効日に対応する現在の階層が表示されます。

組織階層は、次のものを使用して作成できます。

- 「組織階層」ウィンドウ

  **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「組織階層の作成」

- 組織階層ダイアグラマ（グラフィカルな階層を作成し、簡単なドラッグ・アンド・ドロップで変更できます)。

  **関連項目：**『アプリケーション・データ・エクスチェンジおよび階層ダイアグラマ・ユーザーズ・ガイド』の「組織または職階の階層への追加」

# 役割を定義する

## ステップ 57役職を定義する

役職は、企業の方針に従って、一般的な役割または特定の役割のいずれかを定義します。従業員の役割は、組織構成とは別に定義されるため、役割にはある程度の柔軟性をもたせて定義されます。

役職名はビジネス・グループに関連付けした「役職」フレックスフィールド体系セグメントの値の一意の組合せです。

役職を定義するときには、適切な追加情報を指定します。

「役職」ウィンドウを使用してください。

**関連項目：**

- 『Oracle HRMS ユーザーズ・ガイド』の「役職の定義」
- 『Oracle HRMS ユーザーズ・ガイド』の「役職と職階に関する追加情報の入力」

## ステップ 58職階採用ステータスを定義する

各職階には、「提示」、「有効」、「確定」、「消去」、「削除」のいずれかの採用ステータスが必要です。これらのシステム採用ステータスにユーザー名を作成できます。また、必要に応じて各システム名に複数のユーザー名を定義できます。

「ユーザー・タイプおよびステータス」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「職階採用ステータスの定義」

## ステップ 59職階を定義する

オラクル人事管理システムでは、職階とは、組織における従業員の役職を示します。職階は、一般に役割が 1 つの組織に固定しているときに使用します。職階ごとに役職を定義することで、職階別に役割を識別できます。

職階名はビジネス・グループにリンクした、「職階」フレックスフィールド体系セグメントの値の一意の組合せです。

職階を定義するときには、適切な追加情報を指定します。

「職階」ウィンドウを使用してください。

**関連項目：**

- 『Oracle HRMS ユーザーズ・ガイド』の「職階の定義」
- 『Oracle HRMS ユーザーズ・ガイド』の「役職と職階に関する追加情報の入力」

## ステップ 60職階の同期プロセスを設定して毎晩実行する

オラクル人事管理システムは、職階の同期プロセスを使用して、デートトラックされない職階表（PER_ALL_POSITIONS_F）を、デートトラック表（HR_ALL_POSITIONS_F）に加えられた変更で更新します。このプロセスを実行すると、本日以前の有効日を持つデートトラックされた変更が、デートトラックされない表に適用されます。したがって、先日付の変更は、有効になるまで適用されません。

職階の同期プロセスを毎晩実行すると、システムは、その日に有効になった職階の変更で表を自動的に更新します。停電やコンピュータの障害によりこのプロセスが中断した場合は、「新規要求の発行」ウィンドウから手動で開始できます。

**警告：再発行間隔が毎晩実行するように設定されていることを確認してください。**

「新規要求の発行」ウィンドウを使用してください。

**関連項目：**『Oracle Applications ユーザーズ・ガイド』の「要求の発行」

## ステップ 61職階階層を作成する

階層に対して職階を体系化することで、詳細な職階のレポート・ラインを表示できます。また、企業のセキュリティ・プロファイル・グループの定義や、職階のキャリア昇格パスを定義する場合も職階階層も使用できます。

職階階層は、次のものを使用して作成できます。

- 「職階階層」ウィンドウ

  **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「職階階層の作成」

- 職階階層ダイアグラマ（グラフィカルな階層を作成し、簡単なドラッグ・アンド・ドロップで変更できます）。

  **関連項目：**『アプリケーション・データ・エクスチェンジおよび階層ダイアグラマ・ユーザーズ・ガイド』の「組織または職階の階層への追加」

# 等級関連情報を定義する

## ステップ 62等級を定義する

等級は、職務間の相対的な格付けを示し、多くの場合は報酬と福利厚生の適格基準として使用されます。

等級名はビジネス・グループに関連付けした「等級」フレックスフィールド体系セグメントの値の一意の組合せです。

従業員アサイメントの一部として、役職や職階に対して有効等級を定義することで、ユーザーが入力する詳細を相互検証できます。

「等級」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「等級の定義」

## ステップ 63等級レートを定義する

等級レートは等級に直接関連付けられている有効な支給レートの表示に通常使用します。等級レートは固定値または値の範囲として表示できます。

等級レートを定義するときは、値を表に設定します。従業員の等級と表の値を使用することで、従業員の給与を制御したり比較できます。

- 従業員のすべてのエレメントの入力パラメータを検証するために、Formula で等級レート値を使用できる。
- 等級レート値は、比較値を算出するために「等級内支給率表示」ウィンドウで、給与を検証するために「給与管理」ウィンドウでそれぞれ使用できる。

「等級レート」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「等級レートの定義」

## ステップ 64支給スケールを定義する

一般に支給スケールは、支払の実績が「支給スケール」、「予定」または「スケール」として定義されている政府、規制下にある企業または労働組合のある企業で使用されます。

この環境では、勤続期間または設定した日付に従業員の給与を自動的に昇給する方式が一般的です。支給スケールを定義するときは、使用する増分値を使用してポイントを定義します。

オラクル人事管理システムには、システムの定義の増加処理があらかじめ設定されています。日付を設定することで、従業員のステップやポイントの値が自動的に増加します。

**注意：**増加処理は、企業の給与規定に合わせて変更できます。

「支給スケール」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「支給スケールの定義」

## ステップ 65ポイント値を定義する

支給スケールの各ポイントにポイント値を定義します。この値はデートトラックされます。

「ポイント値」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「ポイント値の定義」

## ステップ 66支給ポイントに等級を関連付ける

等級ごとに有効なポイントを数値を使用して定義します。

定義したステップは自動増加処理で使用されます。この処理を使用することで、等級に対して定義した上限値の範囲内で従業員の等級ポイントが増加します。定義した等級上限値より大きいポイントを入力するには、「等級ステップ設定」ウィンドウを使用します。

「等級スケール」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「支給ポイントと等級の関連付け」

# 給与情報を定義する

従業員に対して変動処理エントリのエレメントを入力する前に、必ず従業員アサイメントに給与を含めてください。変動処理エントリの入力は、1 回の給与期間に対してのみ有効です。

## ステップ 67支払方法を定義する

現金や直接振込などの標準カテゴリの支払方法は、システムにあらかじめ定義されています。このような支払方法の名称は自由に定義できます。また Oracle Payroll を導入している場合は、支払方法を使用して従業員への支払を制御できます。

「組織の支払方法」ウィンドウを使用してください。

**関連項目：**「支払方法の定義」

## ステップ 68連結セットを定義する

ビジネス・グループを定義すると、システムは、デフォルトの連結セットを自動的に生成します。Oracle Payroll を導入していない場合、このステップは省略してかまいません。

連結セットは、異なる給与計算の結果を 1 つのレポート・セットに収集したり、他のシステムに転送したりする Oracle Payroll で使用されます。追加の連結セットは、いくつでも定義できます。

「連結セット」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「連結セットの定義」

## ステップ 69給与を定義する

処理や支払の条件に合せて給与グループを定義します。たとえば、半月払給与と週払給与がある場合に、週払給与は工場所在地ごとに管理、処理しなければならないとします。この場合は、1 つの半月給計算と、2 つの週給計算を工場ごとに 1 つずつ、合計 2 つ定義します。

---

**注意：** Oracle Human Resources では、給与カレンダは予算計画カレンダとは異なります。予算計画カレンダは従業員数や予算計画のために定義します。

---

「給与」ウィンドウを使用してください。

**関連項目：**「給与計算の定義」

# 報酬および福利厚生

オラクル人事管理システムでは、エレメントを使用してすべてのタイプの支給項目、控除項目および福利厚生を表示します。エレメントには、報酬や福利厚生の管理に必要な情報が保持されます。

---

**注意：**Standard Benefits または Oracle Advanced Benefits を使用して福利厚生プランを設定する場合は、この項をスキップして、「**総合報酬**」の項にリストされている導入ステップを実行してください。

---

**関連項目：** 2-1 ページ「導入ステップ」

エレメントを定義する前に、定義内容や適格ルールをすべて決めてください。

従業員のエントリ履歴の詳細をロードする場合は、初期設定のデフォルトとして 1951 年 1 月 1 日のような固定の日付を検討してください。こうすることでデータ入力が簡素化されます。

## 入力パラメータ検証を定義する

### ステップ 69参照タイプおよび値を定義する

文字データ型を持つエレメント入力パラメータを検証する値リストを追加作成するには、新規の参照タイプを定義します。

---

**注意：**参照タイプを使用して、フレックスフィールド・セグメントも検証できます。値セットの表検証オプションを使用し、リストのソースとして参照表を使用します。

---

随時新しい参照値を追加できます。値の有効フラグを「No」に設定すると、その値は値リストに表示されなくなります。開始日と最終日を使用して、値をリストに表示する時期を制御できます。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 70ユーザー表を定義する

オラクル人事管理システムでは、設定できる「ユーザー定義表」の数に制限はありません。ユーザー定義表とは、同じ行に異なる値を保持する列から構成される「マトリックス」です。この情報にアクセスするには、Formula で GET_TABLE_VALUE 関数を使用します。

たとえば、役職グループごとに、組合支給レート、控除項目、福利厚生レベルをまとめた 1 つの表を設定するとします。この場合、行に「役職グループ」を、列に各役職グループの値をそれぞれ入力します。

行の正確な値または値範囲を定義できます。

「表体系」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「ユーザー表、列および行の設定」

## ステップ 71表数値を定義する

表数値を定義する必要があります。

「表数値」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「表数値の入力」

## ステップ 72エレメント検証 Formula を定義する

入力パラメータを定義するとき、入力パラメータに対するすべての入力を検証するための Formula を使用できます。

> **注意：**エレメント入力パラメータを定義する前に Formula を定義する必要があります。

Formula のタイプは、次の制約のある「エレメント・パラメータ入力の検証」です。

- Formula には 1 つの入力のみ。
- Formula は、次のような成功またはエラーを示すシステム定義のステータス・コードを返す。

```
FORMULA_STATUS =  'S'  または  'E'
```

- ユーザーに、次のようなメッセージを返すことも可能。リターン・メッセージはメッセージ・ウィンドウに表示される。

```
FORMULA_MESSAGE = ' ...  '
```

「Formula」ウィンドウを使用してください。

**関連項目：**『Oracle Fast Formula ユーザーズ・ガイド』の「検証用 Formula の記述」

# 報酬および福利厚生を定義する

## ステップ 73エレメントおよび入力パラメータを定義する

エレメントとは、あらゆる報酬と福利厚生の基本コンポーネントを表します。また、エレメントを使用して、従業員に貸与した備品を含む現物支給項目を表示できます。

各エレメントは、次のように設定できます。

- 15 個までの入力パラメータを定義できる。
- 値ごとに検証オプションを設定する。
  - 固定
  - 範囲
  - 参照を使用した値リスト
  - Formula
- ホットまたはコールドのデフォルト・ルールを設定する。

給与計算処理にエレメントを使用している場合は、次のことも可能です。

- エレメントに 1 つの入力パラメータ「結果値」を作成する。

**注意：**給与計算処理フラグを「Yes」に設定すると、自動的に結果値が設定されます。

このタイプのエレメントを給与計算で処理する場合は、このフラグを「Yes」に設定する必要があります。

- エレメントが挿入する個々のバランスを変更するには「バランス挿入」ウィンドウを使用する。

「エレメント」ウィンドウを使用してください。

**関連項目：**「エレメントの定義」

## ステップ 74エレメント・リンクを定義する

従業員に対して値を設定できるのは、従業員が定義したエレメントの値を受け取る対象となる場合だけです。従業員がエレメントの対象となるのは、アサイメント詳細とリンク詳細が一致したときです。

エレメントは、組織、グループ、等級、役職、職階、給与、事業所、雇用カテゴリ、給与ベースを組合せてリンクできます。

「エレメント・リンク」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「エレメント・リンクの定義」

## ステップ 75事前定義エレメントを有効にする

オラクル人事管理システムをインストールすると、多数の事前定義エレメントがインストールされます。これらのエレメントは、給与計算で処理される法定控除を表します。

Oracle Payroll をインストールした場合は、これらの控除の処理に必要な Formula とバランスもすべてインストールされます。Oracle Payroll をインストールしていない場合でも、これらのエレメントを使用して情報を記録し、独自の給与システムに転送できます。

これらの事前定義エレメントを有効にするために必要なのは、それらのエレメントのリンクを定義することだけです。

「エレメント・リンク」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「エレメント・リンクの定義」

## 英国での導入のみ

英国の法定控除を表すエレメントの例として、PAYE、NI、PAYE、裁判所命令、EAS 裁判所命令（スコットランドにのみ適用）および CMA 裁判所命令（スコットランドにのみ適用）があります。

英国で導入し、従業員の「税」ウィンドウを使用して PAYE と NI の詳細を入力する場合は、PAYE エレメントと NI エレメントのすべての給与計算に「標準」リンクを定義する必要があります。

PAYE エレメントと NI エレメントのリンクを定義するときは、いくつかのデフォルト入力パラメータを指定する必要があります。次のデフォルトを使用することをお薦めします。

- **税コード** 'BR'
- **税ベース** 'Cumulative'
- **NI カテゴリ** 'A' または 'D'（従業員の大多数が企業年金制度に登録されている場合は、カテゴリ D を使用します）

# 給与のバランス、Formula および結果を定義する

Oracle Payroll をインストールしていない場合、これらのウィンドウはメニューに表示されません。これらのステップをスキップして次の項に進んでください。

## ステップ 76ユーザー・バランスを定義する

**注意：**Oracle Payroll では、総支給額と差引支給額のバランス計算に関するすべての法定要件をサポートするために、多くの事前定義バランスがシステムにインストールされています。給与処理の整合性を保つために、これらのバランスは変更できません。

他のバランスを定義できます。たとえば、特別バランスを定義して、固定処理控除の「停止ルール」を計算する必要があります。遡及支払の計算のための特別バランスも定義する必要があります。

給与バランスを定義するときは、挿入と累計期間を指定する必要があります。

「バランス」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「ユーザー・バランスの定義」

## ステップ 77給与計算 Formula を記述する

給与計算時に処理するエレメントごとに Formula を記述します。Formula タイプは 'Oracle Payroll' です。

**警告：Formula 定義はデートトラックされることに注意してください。給与計算で Formula を使用した後は、必ず変更を Formula に「更新」する必要があります。**

これによって、遡及支給項目または遡及控除項目の再計算について Formula の履歴が保持されます。

「Formula」ウィンドウを使用してください。

**関連項目：**『Oracle Fast Formula ユーザーズ・ガイド』の「検証用 Formula の記述」

### ステップ 78 Formula 実行結果ルールを定義する

給与計算でエレメントを処理すると、システムは Formula を使用して結果を計算します。Formula の結果は、Formula の最後の *Return* 文に含める値です。実行結果ルールは、Formula によって生成された各結果に対して何が行われるかを定義します。

1 つの Formula に異なる結果をいくつでも計算できます。結果のタイプとして、次のものがあります。

- 直接
- 間接
- メッセージ
- 固定処理停止
- 固定処理更新

給与計算には直接結果が 1 つしかありません。これは、通常はエントリの結果値です。

**警告 : ユーザーに支給項目または控除項目タイプの結果値の入力を許可すると、その値は支払の直接結果を提供する Formula 計算を上書きします。**

「Formula 実行結果ルール」ウィンドウを使用してください。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「Formula 実行結果ルールの定義」

## 給与管理

給与管理機能を使用して、個々の従業員の基本報酬バランスを管理します。

### ステップ 79給与エレメントを作成または決定する

企業の給与ベースごとに少なくとも 1 つの給与エレメントが必要です。

ローカライズされた製品に事前定義エレメントが存在する場合は、これらを使用できます。ローカライズされた製品に事前定義エレメントが含まれていない場合、または事前定義エレメントが不十分あるいは不適切である場合は、給与値を格納する新しいエレメントを作成する必要があります。

**注意 :** 管理するさまざまな給与ベースに対して何種類のエレメントを定義する必要があるかを検討してください。給与を年俸として管理し、その値を月額として保存できます。

「エレメント」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「給与エレメントの作成」

## ステップ 80給与エレメントをリンクする

給与エレメントを従業員アサイメントのコンポーネントにリンクして、従業員がエレメントに対して適格となる基準を設定します。

「エレメント・リンク」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「給与エレメントのリンク」

## ステップ 81給与ベースを定義する

給与管理に使用する各給与エレメントの給与ベースを定義します。これにより、時給、月次、年次などの給与の期間タイプが決まります。

「給与ベース」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「給与ベースの定義」

## ステップ 82給与コンポーネントをレビューまたは作成する

参照タイプ PROPOSAL_REASON の値として事前定義されている給与コンポーネントをレビューします。必要に応じて、独自の給与コンポーネントを作成します。

新しいコンポーネントを「給与管理」フォルダに表示するには、ビューも変更する必要があります。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「給与コンポーネントの作成」

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 83業績考課レート・タイプを定義する

「抜群」、「優秀」、「平均」などの業績考課レートを記録する場合は、「アプリケーション・ユーティリティ参照」ウィンドウで参照タイプ PERFORMANCE_RATING の値として業績考課レートを入力します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

### ステップ 84給与管理承認機能を追加する

給与条件提示を承認する職責のメニューに、機能「給与管理承認」を追加します。

**注意：**この機能が職責に存在しない場合、ユーザーは給与条件提示を入力できますが、承認はできません。

「メニュー」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「メニューの定義」

### ステップ 85給与入力を検証する

給与入力は、次のどちらかの方法で検証（妥当性を検査）できます。

- 従業員の等級に対して定義された有効範囲から外れる給与条件提示が入力された場合に、ユーザーに警告します。この方法では、等級レート範囲を使用します。
- 有効範囲を外れる給与や、Formula により実行された検証結果が無効である給与をユーザーが承認できないようにします。ただし、給与条件提示を承認しようとする時点まで、この検証処理は実行されないことに注意してください。この方法では、エレメントの入力パラメータ検証を使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「給与入力の検証」

## 不就業管理と有給休暇付与 (PTO)

従業員に対して、休暇や病欠に使用する有給休暇を付与するため、必要なだけプランを設定できます。それぞれのプランには時間数または日数の単位が用意され、異なるアサイメント状況を持つ従業員の有給休暇の頻度、上限、繰越、開始日、残余日数などに関するルールを用意できます。

### 不就業管理の設定

### ステップ 86変動処理不就業エレメントを定義する

それぞれの有給休暇付与プランに対して、変動処理エレメントを定義し、休暇や病欠に使用した実時間を記録するために値を入力します。

「エレメント」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「不就業エレメントの定義およびリンク」

## ステップ 87不就業エレメントをリンクする

各不就業エレメントをリンクして、その種類の不就業の適格者を定義します。

「エレメント・リンク」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「エレメント・リンクの定義」

## ステップ 88不就業タイプのカテゴリを定義する

不就業タイプのカテゴリを参照タイプ ABSENCE_CATEGORY の値として定義し、不就業理由を参照タイプ ABSENCE_REASON の値として定義します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 89不就業タイプを定義して不就業エレメントに関連付ける

各不就業タイプを定義し、それを不就業エレメントに関連付けます。

「不就業タイプ」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「不就業タイプの定義」

## ステップ 90エレメント・エントリの初期値を設定する

減少バランスを設定した不就業タイプの場合は、「エレメント・エントリ」ウィンドウまたは「MIX バッチ」機能を使用して、不就業タイプに適格の従業員に対してエレメント・エントリの初期値を設定します。

個々の従業員について入力する場合は、『Oracle HRMS ユーザーズ・ガイド』の「手動エレメント・エントリの作成」を参照してください。バッチで入力する場合は、『Oracle HRMS ユーザーズ・ガイド』の「BEE を使用してバッチ・エレメント・エントリの作成」を参照してください。

## 有給休暇付与プランの設定

### ステップ 91有給休暇付与プランに対して不就業タイプのエレメントを定義、リンクする

不就業エレメントを定義、リンクしていない場合は、これを行います。

「エレメント」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「不就業エレメントの定義およびリンク」

### ステップ 92プランの不就業タイプを定義する

「不就業詳細」ウィンドウを使用して有給休暇の取得日数を入力するには、プランに不就業タイプを設定し、その不就業タイプと不就業エレメントを関連付けます。

「不就業タイプ」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「不就業タイプの定義」

### ステップ 93新規起算日ルールを定義する

シードされた開始ルールには、入社日、カレンダ年の始めおよび入社日の 6 か月後の 3 つがあります。他のルールが必要な場合は、参照タイプ US_ACCRUAL_START_TYPE の値として定義します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

### ステップ 94有給休暇と繰越の Formula を決定する

使用する有給休暇と繰越の Formula を決定します。シードされた Formula を使用、カスタマイズまたは独自の Formula を記述できます。

「Formula」ウィンドウを使用してください。

> **関連項目：**『Oracle Fast Formula ユーザーズ・ガイド』の「有給休暇付与プランの Formula の記述」

## ステップ 95不適格 Formula を記述する

有給休暇 Formula で不適格期間を定義している場合に、バッチ・エレメント・エントリ（BEE）を使用して有給休暇付与プランに不就業を入力するには、不適格 Formula を定義します。BEE は、この Formula をコールして、従業員が累積有給休暇を使用できるかどうかをチェックします。

---

**注意：**シードされた有給休暇 Formula を使用する場合は、不適格 Formula を定義する必要はありません。有給休暇 Formula では、「有給休暇付与プラン」フォームに入力された不適格期間が使用され、BEE 検証で同じ値を使用できます。

---

「Formula」ウィンドウを使用してください。

**関連項目：**『Oracle Fast Formula ユーザーズ・ガイド』の「有給休暇付与プランの Formula の記述」

## ステップ 96新しい有給休暇カテゴリを定義する

シードされた有給休暇カテゴリがいくつかあります。追加のカテゴリが必要な場合は、参照タイプ US_PTO_ACCRUAL の値として定義します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 97有給休暇付与プランを定義する

有給休暇付与プランを定義し、そのプランで使用する Formula と不就業エレメントを選択します。

「有給休暇付与プラン」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「有給休暇付与プランの定義」

## ステップ 98勤続期間の範囲を設定する

オプションとして、有給休暇付与プランの基準となる勤続期間の範囲を設定します。

「有給休暇付与期間」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「勤続期間の範囲設定」

### ステップ 99計算ルールをレビューする

プランの計算ルールをレビューします。必要に応じて、追加エレメントを作成し、「計算ルール」ウィンドウで選択することによって、そのエレメントをプランに関連付けます。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「有給休暇の残余日数計算の変更」

## エレメント・セット

### ステップ 100エレメント・セットを定義する

オラクル人事管理システムでは、次の目的にエレメント・セットを定義できます。

- フォーム・カスタマイゼーションを使用して、エレメントへのアクセスを制限する。
- エレメント・セットの分配セットに原価を分配する。
- 給与計算で制限付きセットを処理する。
- BEE（バッチ・エレメント・エントリ）を使用して、制限付きセットの値を入力する。

エレメント・セットは、給与や、給与と賞与など名称を付けてエレメントのリストとして定義します。また、エレメント・セットは分類を使用して定義できます。たとえば、給与・支給項目のエレメントすべてに対するアクセスを制限できます。

「エレメント・セットおよび分配セット」ウィンドウを使用してください。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「エレメント・セットまたは分配セットの定義」

# 総合報酬

Standard Benefits と Oracle Advanced Benefits では、導入ステップの多くが共通します。以下では、Oracle Advanced Benefits にのみ適用される導入ステップについて説明します。

## 「福利厚生」タブ・リージョン

セキュリティ保護された職責を使用してこのリージョンの福利厚生情報にアクセスできるようにするには、「個人情報」ウィンドウに福利厚生のタブ・リージョンを追加する必要があります。このリージョンには、個人が所属する福利厚生グループおよび医療プランなどの情報が含まれます。

### ステップ 101「福利厚生」タブ・リージョンを「個人情報」ウィンドウに追加する

システム管理責任者またはアプリケーション開発責任者は、「メニュー」ウィンドウを使用して、福利厚生の別リージョンを「個人情報」ウィンドウに追加できます。

1. 「メニュー」フィールドで BEN_MANAGER メニューを問い合せます。
2. 行を追加して、「機能」フィールドで「HR 福利厚生の表示」を選択します。
3. 作業内容を保存します。

「メニュー」ウィンドウを使用します。

**関連項目：**『Oracle Applications システム管理者ガイド』の「「メニュー」ウィンドウ」

## 福利厚生の適格

福利厚生の適格を判別するには、加入者の適格プロファイルを定義します。福利厚生の拠出金および分配金の変数レートを決定する場合は、適格要因も使用できます。

### ステップ 102福利厚生グループを定義する

福利厚生グループは、福利厚生または標準のアクティビティ・レートが適用されるかされないかによって区分される個人のカテゴリとして定義できます。福利厚生グループは、適格プロファイルまたは変数レート・プロファイルのオプション・コンポーネントの 1 つです。

「福利厚生グループ」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「福利厚生グループの定義」

### ステップ 103郵便番号域を定義する

福利厚生の適格の制限やアクティビティ・レートの変更を、個人の住所に基づいて行う場合は、郵便番号域を定義します。

郵便番号域は、サービス区域のコンポーネントでもあります。

「郵便番号域」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「郵便番号域の定義」

### ステップ 104サービス区域を定義する

住所の郵便番号に従って、個人をグループ別にまとめるには、サービス区域を定義します。サービス区域は、適格プロファイルまたは変数レート・プロファイルのオプション・コンポーネントの 1 つです。

「サービス区域」ウィンドウを使用する

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「サービス区域の定義」

### ステップ 105規約を定義する

規約とは、1 つ以上の福利厚生の管理について行政または政府が定義した条例、方針、要件のいずれかとして定義されます。

「規約」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「規約の定義」

## 導出適格要因

導出要因はシステムの計算した値で、福利厚生に対する適格を定義したりアクティビティ・レートを定義したりするのに使用できます。

### ステップ 106報酬レベルに対する導出要因を定義する

報酬レベル要因を定義して、個人の所得申告または指定したバランス・タイプを基に個人の報酬レベルを導出する方法を決めます。

「導出要因」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : 報酬レベル」

## ステップ 107フルタイム雇用のパーセントに対する導出要因を定義する

フルタイム % 要因を定義して、個人のフルタイム雇用のパーセントを導出する方法を決めます。

「導出要因」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : フルタイム雇用に対するパーセント」

## ステップ 108期間内の労働時間に対する導出要因を定義する

期間内の労働時間要因を定義して、一定期間における個人の労働時間を導出する方法を決めます。

「導出要因」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : 期間内の労働時間」

## ステップ 109年齢要因を定義する

年齢要因を定義して、個人の年齢を導出する方法を決めます。

「導出要因」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : 年齢」

## ステップ 110勤続期間要因を定義する

勤続期間要因を定義して、個人の勤続期間の算出方法を決めます。

「導出要因」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : 勤続期間」

## ステップ 111年齢および勤続要因を定義する

年齢および勤続期間要因を定義して、年齢要因と勤続期間要因を組み合わせます。

「導出要因」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「導出要因の定義 : 年齢および勤続期間」

# 適格プロファイル

### ステップ 112適格プロファイルを定義する

適格プロファイルの定義は、福利厚生の適格要件を導入する基本的な方法です。適格プロファイルを報酬オブジェクト（定義した福利厚生）にリンクして、適格プロセスが実行されたときに、適格プロファイルの要件を満たす個人だけが福利厚生の対象になるようにします。

「加入者適格プロファイル」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「適格プロファイルの定義」

### ステップ 113扶養家族補償範囲適格プロファイルを定義する

扶養家族補償範囲適格プロファイルを定義して、扶養家族に適格要件を適用します。

「扶養家族補償範囲適格プロファイル」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「扶養家族補償範囲適格プロファイルの定義」

# ライフ・イベントを定義する（Oracle Advanced Benefits）

登録処理を実行するきっかけとなりうる個人レコードの変更として、ライフ・イベントを定義します。ライフ・イベントは、結婚や扶養家族の誕生といった仕事に関係のないものと、職務の変更といった仕事と直接関係のあるものがあります。予定登録もライフ・イベントの 1 つです。

### ステップ 114ライフ・イベント処理を定義する

選択可能性、アクティビティ・レートおよび補償範囲レベル、補償範囲の日付、連絡方法、自動およびデフォルト登録処理を制御するには、ライフ・イベントを定義します。

「ライフ・イベント理由」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「ライフ・イベント理由の一般特性の定義」

### ステップ 115個人変更を定義する

個人変更を示すデータベース・フィールドの値を指定して、ライフ・イベントを起動する個人レコードの変更を定義します。

「個人変更」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「個人変更の定義」

### ステップ116個人変更とライフ・イベントを関連付ける

定義したライフ・イベントごとに、そのライフ・イベントを起動する個人変更を関連付けます。

「個人変更連携ライフ・イベント」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「個人変更とライフ・イベントの関連付け」

### ステップ117関連個人変更を定義する

ライフ・イベントを起動する個人レコードの変更を定義するには、関連個人変更を示すデータベース・フィールドの値を指定します。

たとえば、退職ライフ・イベントを定義して、退職従業員の福利厚生の補償範囲を終了させることが可能です。その場合、プライマリ加入者が退職したときに、その扶養家族の補償範囲を終了する個人のライフ・イベントもあわせて定義できます。

「関連個人変更」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「個人変更の定義」

### ステップ118関連個人変更とライフ・イベントを関連付ける

関連個人変更ごとに、これを定義した個人のライフ・イベントに対して関連付けます。関連個人変更は、プライマリ加入者の人事システム・レコードに対する変更であり、プライマリ加入者に関連する個人のライフ・イベントが生成されることもあります。

「個人変更連携ライフ・イベント」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「個人変更とライフ・イベントの関連付け」

## プログラム設定

従業員または適格加入者に提供される福利厚生として報酬オブジェクトを定義します。

報酬オブジェクトは、報酬オブジェクトの階層に従って、次のように配置されます。

- プログラム
- プラン・タイプ
- プラン
- オプション

プログラム・レベルで設定した定義は、下の階層レベルで定義を上書きしない限り、そのプログラムのプラン・タイプ、プランおよびオプションに適用されます。

## ステップ 119精算可能な物品サービス・タイプを定義する

精算として承認する物品サービスを定義します。次に、その精算プランに対して、1 つ以上の物品サービス・タイプを関連付けます。

「物品サービス」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「精算可能な物品サービス・タイプの定義」

## ステップ 120プログラムまたはプラン年期を定義する

プログラムまたはプラン年期を定義して、福利厚生プログラムまたはプランの期間に補償範囲境界を設定します。

「プログラム / プラン年数」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プログラムまたはプラン年期の定義」

## ステップ 121プラン・タイプを定義する

プラン・タイプを定義すると、医療プラン、貯蓄プランといった一般的な福利厚生をカテゴリに分類できます。

「プラン・タイプ」ウィンドウを使用する

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プラン・タイプの定義」

## ステップ 122オプションを定義する

オプションを定義して、プランで利用できる補償範囲レベルを示したり、貯蓄プランの投資オプションを定義します。

「オプション」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「オプションの定義」

## ステップ 123プランを定義する

プランは、適格加入者が登録できる福利厚生です。一般的なプランとして、医療プラン、団体生命保険プラン、株式購入プランなどがあります。

「プラン」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「福利厚生プランの定義」

### ステップ 124精算プランを定義する

精算プランを使用すると、適格加入者が購入可能な物品サービスを定義できます。加入者は、立て替えた物品サービスの費用に対して精算要求を発行できます。

「プラン精算」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「一般プラン精算情報の定義」

### ステップ 125プログラムを定義する

プログラムを定義して、パッケージとして提供する福利厚生をグループ化できます。一般に、プログラムは、プラン・タイプ、プラン、オプションから構成されます。

「プログラム」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プログラムの一般特性の定義」

## 登録要件

登録要件を定義して、適格個人が福利厚生に登録できる時期を制御します。

### ステップ 126プログラム登録要件を定義する

登録要件は、適格加入者をプログラムに登録する方法を決めます。

Standard Benefits を使用している場合は、無制限登録タイプに基づいて登録要件を定義します。Oracle Advanced Benefits の福利厚生を使用している場合は、プログラムに自動またはデフォルトの登録ルールを適用するかどうかを指定できます。

「プログラム登録要件」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プログラムの登録方法の定義」

### ステップ 127プランの登録要件を定義する

「プラン登録要件」ウィンドウを使用して、プログラム外のプランまたはプラン内のオプションに対する登録要件を定義します。また、このウィンドウを使用して、受取人の指定要件を定義することもできます。

「プラン登録要件」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プランの登録方法の定義」

# アクティビティ・レート計算および補償範囲計算

アクティビティ・レート計算によって、福利厚生の利用に必要な拠出金および分配金がある場合は分配金のレートが決まります。

## ステップ 128変数アクティビティ・レートを計算する

報酬オブジェクトのアクティビティ・レートが加入者ごとに異なる場合は、変数アクティビティ・レート計算を定義します。

「変数レート・プロファイル」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「変数レート・プロファイルの一般情報の定義」

## ステップ 129補償範囲を計算する

補償範囲オプションのあるプランの場合は、福利厚生プランで使用可能な補償範囲を定義します。補償範囲計算には、計算結果に関係なく、補償範囲レベルとして使用可能な最大値および最小値を使用することが可能です。Oracle Advanced Benefits ユーザーの場合は、補償範囲レベルはライフ・イベントに従って変化します。

「補償範囲」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「プランの補償範囲計算の定義」

## ステップ 130複数プラン・タイプ間で補償範囲制限を定義する

加入者がプログラムの複数のプラン・タイプ間で選択できる補償範囲の最大値および最小値を定義します。

「複数プラン・タイプ間の補償範囲」ウィンドウを使用します。

**関連項目：**

- 複数プラン・タイプ間での補償範囲制限
- 複数プラン・タイプ間での補償範囲制限の定義、オラクル人事管理システムを使用した報酬と福利厚生の管理

## ステップ 131実掛金原価を計算する

プラン主催者が福利厚生仕入先に支払い義務のある実掛金原価を計算する基準を保守する必要があります。

「実掛金」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「実掛金の定義」

### ステップ 132期間累計限度を定義する

プランまたはプラン内のオプションで加入者が年間に拠出する金額が制限されている場合は、拠出金の期間累計限度を定義します。標準の拠出金を定義すると、拠出金の制限が必要なプランまたはプラン内のオプションに対して、期間累計限度を関連付けることができます。

「期間累計限度」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「期間累計限度の定義」

### ステップ 133標準の拠出金 / 分配金のアクティビティ・レートを定義する

標準のアクティビティ・レート計算を定義して、福利厚生の拠出金または分配金の計算を行います。

**注意：**これに先立って、関連するエレメントは作成しておく必要があります。

「標準の拠出金 / 分配金」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「標準の拠出金 / 分配金のアクティビティ・レートの定義」

## レポート・グループ

### ステップ 134レポート・グループを定義する

レポート・グループを定義して、そのレポート・グループに対して 1 つ以上のプログラムおよびプランをリンクすることが可能です。レポート・グループのレポートを実行すると、レポート・グループに定義されたプログラムおよびプランに基づいて、レポート結果が生成されます。

また、調整機関やレポート・グループを管理する規約も定義できます。

「レポート・グループ」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「レポート・グループの定義」

# フレックス・クレジット計算（Oracle Advanced Benefits）

## ステップ 135福利厚生資金の特性を定義する

福利厚生資金を定義して、加入者がフレックス・クレジットを利用する方法や、超過フレックス・クレジットをロールオーバー、換金、喪失する方法を定義します。

「福利厚生資金」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「福利厚生資金の一般特性の定義」

## ステップ 136フレックス・クレジットを定義する

フレックス・クレジット計算を定義して、報酬オブジェクトに計算をリンクします。フレックス・クレジットにリンクする報酬オブジェクトは、フレックス・クレジットを定義するレベルに関係なく、プログラムの一部である必要があります。

「フレックス・クレジット定義」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「フレックス・クレジットの定義」

## ステップ 137連絡を定義する

従業員および潜在的加入者への連絡を定義します。連絡を開始する条件、連絡方法、連絡メディアを定義します。

「連絡タイプ」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「連絡の定義」

# 管理

## ステップ 138福利厚生バランスを定義する

福利厚生バランスは、法的システム・データをオラクル人事管理システムに転送する場合に役立ちます。福利厚生バランスのタイプを定義し、「個人福利厚生バランス」ウィンドウを使用して特定の個人のタイプに値を割り当てます。

「福利厚生バランス」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「福利厚生バランス・タイプの定義」

# オンライン福利厚生サービスの定義

「オンライン福利厚生サービス」ウィンドウを使用して、中央ローケーションからさまざまな福利厚生ウィンドウにアクセスします。このウィンドウからアクセス可能なウィンドウを構成し、定義済みのユーザー・イベントに従って表示されるポップアップ・メッセージを定義することが可能です。

## ステップ 139デスクトップ・アクティビティ・リストを保守する

「オンライン・アクティビティの保守」ウィンドウを使用して、「オンライン福利厚生サービス」ウィンドウの「デスクトップ・アクティビティ」リストで使用できる機能およびウィンドウを定義します。

「オンライン・アクティビティの保守」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「オンライン・アクティビティの保守」

## ステップ 140ポップアップ・メッセージを保守する

定義済みのユーザー・イベントに従って「オンライン福利厚生サービス」ウィンドウに表示されるメッセージを構成できます。「メッセージ」ウィンドウでメッセージ・テキストを作成します。

「ポップアップ・メッセージの保守」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「ポップアップ・メッセージの保守」

# 個人情報およびアサイメント

オラクル人事管理システムでは、ユーザーが自由に名称を定義でき、システム内の Person タイプや各アサイメントの従業員ステータスを識別する名称として使用できます。

## Person タイプとアサイメント状況

### ステップ 141 Person タイプを定義する

システム内の Person タイプを識別する名称を自由に定義できます。

**注意 :** Person タイプはフォーム・カスタマイゼーションの共通オプションです。

「Person タイプ」ウィンドウを使用します。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「Person タイプの定義」

### ステップ 142 従業員のアサイメント状況を定義する

オラクル人事管理システムでは、独自に名称を設定してアサイメント別に従業員ステータスを表示できます。たとえば、従業員が一時的に別のロールに割り当てられている間、「休止」になっているアサイメントを識別する独自のステータスを定義します。

「アサイメント状況」ウィンドウを使用します。

**関連項目 :** 『Oracle HRMS ユーザーズ・ガイド』の「アサイメント状況の定義」

### ステップ 143契約ステータスを作成する

契約ステータスは 250 個まで作成できます。「契約」ウィンドウで契約ステータスを選択します。CONTRACT_STATUS 参照を使用して必要な契約ステータスを作成してください。

契約ステータスにはプリフィクスを含めることができ、その契約が有効、無効または不要になった古い契約であるかを定義できます。

- **A-**: 契約が有効であることを示す場合にこのプリフィクスを使用します。
- **O-**: 不要になった古い契約であることを示す場合にこのプリフィクスを使用します。

**注意 :** 契約ステータスにプリフィクスがない場合は、その契約が無効なことを示すとみなされます。

「アプリケーション・ユーティリティ参照」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## 特別個人情報（「個人特別情報」キー・フレックスフィールド体系）

「個人特別情報」キー・フレックスフィールドを使用して、個人情報やアサイメントに含めない特別情報を入力します。それぞれの情報は、フレックスフィールドの個別の体系として定義されます。たとえば、医療情報を保持するための体系を設定したりできます。

このフレックスフィールドは次の領域で使用します。

- 個人の特別情報詳細
- 役職と職階の要件の照合

オラクル人事管理システムに設定する特別情報タイプごとに、「個人特別情報」フレックスフィールド体系を設計する必要があります。各体系に対して次の項目を含める必要があります。

- 体系名とセグメント数
- フレックスフィールドのセグメントの名称、順序、検証オプション
- 使用するフレックスフィールドの値セットと値リスト

フレックスフィールド体系の定義はシステム管理者が設定します。

**注意：**「キー・フレックスフィールド・データベース・アイテムの作成」プロセスを使用して、「個人特別情報」フレックスフィールド体系のセグメントにデータベース・アイテムを作成することはできません。

### ステップ 144「個人特別情報」フレックスフィールドの値セットを定義する

ユーザーが入力できるセグメントの値を検証するには、特定の値セットを定義する必要があります。

値セットの属性を使用して、入力できる値のタイプや、各セグメントに入力できる文字数を制御します。値セットの属性は、値の検証方法も制御します。

値セットは、同一フレックスフィールドの異なるセグメントや他のフレックスフィールドのセグメントで共有できます。

「値セット」ウィンドウを使用します。

**関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「値セットの定義」

## ステップ 145「個人特別情報」フレックスフィールドのセグメントを定義する

使用するセグメントを含む「個人特別情報」フレックスフィールド体系を定義します。この体系は「特別情報タイプ」ウィンドウで詳細を入力するのに使用します。

「キー・フレックスフィールド・セグメント」ウィンドウの「動的挿入許可」チェック・ボックスをチェックします。「No」に設定すると、「特別情報タイプ」ウィンドウに新規詳細を入力できません。

---

**注意 :** セグメントの検証に値セットを使用する必要はありません。値セットを指定しない場合は、最大 150 文字までの英数字を自由に入力できます。

---

「キー・フレックスフィールド・セグメント」ウィンドウを使用します。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 146「個人特別情報」フレックスフィールドのセグメントの値を定義する

「個人特別情報」フレックスフィールドで使用する値セットに「独立」または「依存」検証を選択した場合は、値セットに有効値リストを定義する必要があります。

「セグメント値」ウィンドウを使用します。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「セグメント値の定義」

## ステップ 147「個人特別情報」フレックスフィールドの相互検証ルールを定義する

ユーザーが入力できるセグメント間の組合せを制御するには、相互検証ルールを定義します。

相互検証ルールは、セグメントの組合せを追加または除外するように定義します。各セグメントには値の上 / 下限値の範囲を定義できます。

「相互検証ルール」ウィンドウを使用します。

> **関連項目 :** 『Oracle Applications フレックスフィールド・ガイド』の「相互検証ルールの定義」

## ステップ 148「個人特別情報」フレックスフィールドの別名を定義する

よく使用するセグメント値の組合せをデフォルト・オプションとして指定する場合は、そのような組合せに別名を定義します。

「短縮別名」ウィンドウを使用します。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「短縮別名の定義」

## ステップ 149「個人特別情報」フレックスフィールド体系を確定、コンパイルする

これでフレックスフィールド定義を確定する準備が整いました。「フレックスフィールドのセグメントの定義」ウィンドウに移動してください。「フレックスフィールド定義の確定」フィールドを「Yes」に設定して、変更内容を保存します。こうすることで、オラクル人事管理システムによって、「個人特別情報」フレックスフィールド定義の確定とコンパイルが開始されます。フレックスフィールド定義をコンパイルすると、すでに定義しているデフォルト、値およびルールが「個人特別情報フレックスフィールド」ウィンドウで使用可能になります。

「キー・フレックスフィールド・セグメント」ウィンドウを使用します。

> **関連項目：**『Oracle Applications フレックスフィールド・ガイド』の「キー・フレックスフィールド体系の定義」

## ステップ 150ビジネス・グループの特別情報タイプを登録する

「個人特別情報」フレックスフィールド体系を定義したら、それをビジネス・グループに関連付けます。

この作業には全表示職責を使用します。

- ビジネス・グループで使用する各情報タイプを選択します。
- 各タイプのカテゴリを選択します。
  - 「役職」は役職要件に使用します。
  - 「職階」は職階要件に使用します。
  - 「スキル」は Oracle Training Administration で使用します。
  - 「その他」は個人特別情報で使用します。
  - 「ADA」は米国でのみ使用します。障害のある従業員の情報を入力するための特別情報タイプです。
  - 「OSHA」は米国でのみ使用します。従業員の勤務中の事故や職業病に関する情報を入力するための特別情報タイプです。

**提案:** カテゴリの「その他」チェック・ボックスをブランクにすると、個人情報を保持する情報タイプは使用できません。しかし、役職または職階に限られた情報タイプを保持する特別情報タイプとして使用できるようになります。

「特別情報タイプ」ウィンドウを使用します。

> **関連項目:** 『Oracle HRMS ユーザーズ・ガイド』の「特別情報タイプの有効化」

# 特定業務機能

## 予算計画

### ステップ 151参照タイプと値を定義する

人数や工数に換算した予算計画の表示単位は、オラクル人事管理システムにシステム定義されています。新しい期間タイプは追加できますが、システム定義の期間タイプは変更したり削除することはできません。

BUDGET_MEASUREMENT_TYPES の値を定義します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用します。

> **関連項目:** 『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

### ステップ 152期間タイプを定義する

一般的な期間タイプは、オラクル人事管理システムにシステム定義されています。システム定義の期間タイプの名称を変更することはできますが、新しいタイプを追加することはできません。

「期間タイプ」ウィンドウを使用します。

> **関連項目:** 「期間タイプの名称の変更」

### ステップ 153予算計画カレンダを定義する

カレンダを使用して予算計画年度を定義します。

「予算計画カレンダ」ウィンドウを使用します。

> **関連項目:** 『Oracle HRMS ユーザーズ・ガイド』の「予算計画カレンダの定義」

### ステップ 154予算計画を定義する

予算計画を定義する際、システムを使用して予算計画の予定配属人数に対する実換算人数（実績値）を確認できます。

実換算人数の入力のないアサイメントは予算計画ではカウントされません。「予算計画値」ウィンドウを使用して、アサイメントの予算計画の単位ごとに実換算人数を入力します。

「予算計画」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「予算計画の定義」

## 評価方式

### ステップ 155評価タイプを定義する

オラクル人事管理システムでは、「難易度」ウィンドウを使用して職階や役職の評価情報の要約を入力できます。

評価方式の名称を参照タイプ EVAL_SYSTEM の値として定義します。

Hay 評価方式や他の評価方式の詳細なスコアを入力するには、「追加評価詳細」付加フレックスフィールドを使用して、このような情報を保持したり検証します。

実施した考課に関するコメントや情報を保持できます。

**注意：**複数の評価方式を使用する場合、評価名に対してコンテキスト依存のセグメントを定義できます。

「アプリケーション・ユーティリティ参照」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

### ステップ 156役職または職階の有効等級を定義する

オラクル人事管理システムでは、役職に有効等級を定義できます。こうすることで、役職や等級の情報を入力するときに「アサイメント」ウィンドウでユーザーに警告メッセージを表示できます。

「有効等級」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「役職または職階に対する有効等級の入力」

# 要件照合

コンピテンスを設定してある場合、これらを役職および職階の要件として入力し、個人のコンピテンス・プロファイルと照合できます。

記録したい役職および職階の要件が他にもあるが、コンピテンスとして定義されていない場合、「個人特別情報」キー・フレックスフィールドを使用して設定できます。それぞれの要件タイプを、フレックスフィールドの 1 例である特別情報タイプとして設定できます。

また、特別情報タイプでは、タイプごとに個人情報の入力を許可するかどうかを選択できます。そのためには、「特別情報タイプ」ウィンドウでカテゴリを選択します。個人情報の入力を認めることで、個人を役職または職階の要件と照合できます。標準レポート（スキル照合）は、システム内にある個人の特別情報詳細と役職の要件とを照合するために用意されています。

## ステップ 157役職の要件を定義する

役職に割り当てた従業員に必要な属性を定義できます。このような属性は、必須条件と、要求事項に分類されます。

必要要件の定義には、特別個人情報に定義した、「個人特別情報」フレックスフィールド体系とセグメントを使用できます。

「役職」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「役職と職階の要件入力」

## ステップ 158職階の要件を定義する

職階を定義した後、その職階に割り当てた従業員ごとに必要な属性を定義できます。このような属性は、必要要件と、要求事項に分類されます。必要要件は、特別個人情報に定義した「個人特別情報」フレックスフィールド体系を基準にします。

「職階」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「役職と職階の要件入力」

# キャリア・マネジメントおよび後任管理

## 採用

### ステップ 159応募者のアサイメント状況を定義する

応募者のアサイメント状況では、採用活動の選考状況を定義できます。

オラクル人事管理システムでは、このような採用活動の選考状況に名称をつけて自由に設定できます。たとえば、一次面接を実施した応募者と、応募時に不合格となった応募者のステータスを独自に定義できます。

このようなユーザー・ステータスを使用して、すべての応募者の選考状況を把握できます。

「アサイメント状況」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「アサイメント状況の定義」

## キャリア・マネジメント

業績考課管理システムの一環としてコンピテンス・アプローチを開発する場合、コンピテンス・アプローチで取り組むべき企業の戦略的目標または目的を設定する必要があります。これにより、目的に合った測定方法の設定、コンピテンスの作成、査定と評価のテンプレートの作成ができます。

Oracle Self-Service Human Resources を使用してセルフ・サービスの人事管理をマネージャや従業員に提供する場合は、追加の導入ステップを実行する必要があります。

**関連項目：**『Oracle Self-Service Human Resources（SSHR）インプリメンテーション・ガイド』の「導入ステップ（SSHR)」

### ステップ 160レート・スケールを作成する

企業のコンピテンスを一般的な形で表現する場合、レート・スケールを作成します。

「レート・スケール」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「レート・スケールの作成」

### ステップ 161コンピテンスを作成する

自社のニーズに最も合致したコンピテンスを作成します。個別方法を使用する場合、作成したコンピテンスそれぞれに熟達度レベルを設定する必要があります。

「コンピテンス」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「コンピテンスの作成」

## ステップ 162コンピテンス・タイプを作成する

欠員募集やレポート作成などのために、関連したコンピテンスをグループ化できます。

参照タイプ COMPETENCE_TYPE を使用して必要なコンピテンス・タイプを作成します。

「アプリケーション・ユーティリティ参照」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「参照タイプおよび値の追加」

## ステップ 163コンピテンスをタイプ別にまとめる

関連するコンピテンスをまとめる必要があります。

「コンピテンス・タイプ」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「コンピテンスのタイプ別グループ化」

## ステップ 164コンピテンス要件を定義する

企業が現在および将来の目標を達成できるよう、コンピテンス要件を定義する必要があります。

「コンピテンス要件」ウィンドウを使用します。

『Oracle HRMS ユーザーズ・ガイド』の「コンピテンス要件の定義 - コアまたは一般コンピテンス」

『Oracle HRMS ユーザーズ・ガイド』の「コンピテンス要件の定義 - コア・コンピテンスなし」

## ステップ 165役職と職階の勤務選択を入力する

社内（または顧客先）に配属する従業員や応募者、請負業者、退職者の能力に影響を及ぼす可能性がある勤務選択を入力できます。勤務選択には、出張や転勤に対する希望、労働時間や勤務スケジュールに対する希望などが含まれます。勤務選択は、役職または職階に対して入力し、個人情報に入力された個人の勤務選択と比較できます。

「勤務選択」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「役職または職階の勤務選択の入力」

## ステップ 166機能を定義する（コンピテンス・アプローチを OTA に導入する）

Oracle Human Resources と Oracle Training Administration が導入されている場合、研修アクティビティに参加することで受講者がコンピテンス、スキルまたはその両者の組合せ（コンピテンスおよびスキル）として取得可能な資格、属性、知識を保持できます。

研修アクティビティを通したコンピテンスの付与へ段階的に移行できるよう、パラメータを使用します。パラメータを使用すると、スキル、コンピテンスまたはその両方をユーザーが「アクティビティ」ウィンドウから入力できるかどうかを指定できます。また、アクティビティを通して習得したコンピテンスを受講者のコンピテンス・プロファイルに直接ユーザーが追加できるようにするのにもパラメータを使用します。

「フォーム機能」ウィンドウを使用します。

**関連項目：**『Oracle Training Administration の使用』の「機能の定義」

## ステップ 167資格タイプを作成する

企業が認める資格タイプはすべて入力できます。

「資格タイプ」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「資格タイプの作成」

## ステップ 168学校および大学を作成する

企業の認める資格の授与元である団体を作成する必要があります。この情報が、資格の取得先を記録する際に使用されます。このような学校や大学を自動的に Oracle Human Resources にロードしなかった場合は、手動で入力できます。

---

**注意：**入力する学校と大学は、作成するビジネス・グループすべてで使用できるので、入力またはロードは一度だけで済みます。

---

「学校と大学」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「学校および大学の作成」

# 査定および評価

### ステップ 169 Oracle Self-Service Human Resources（SSHR）を導入する

評価と査定のテンプレートを作成する前に、Oracle Workflow を使用して SSHR の Web 処理を設定するなど、追加の SSHR 導入タスクを実行する必要があります。

**関連項目：**『Oracle Self-Service Human Resources（SSHR）インプリメンテーション・ガイド』の「導入ステップ（SSHR）」

### ステップ 170査定テンプレートを作成する

企業内で行われるさまざまな評価すべてに査定テンプレートを作成できます。

「査定テンプレート」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「査定テンプレートの作成」

### ステップ 171評価テンプレートを作成する

評価担当者にインストラクションを与え、どの質問がどの評価のものか指定し、使用すべき業績考課レートを指定するために、評価テンプレートを作成します。

あらかじめ用意されている評価テンプレートのサンプルを 1 つ選んでそれを必要に合わせて修正するか、独自の評価テンプレートを作成できます。

「評価テンプレート」ウィンドウを使用します。

**関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「評価テンプレートの作成または変更」

# キャリア計画と後任計画

Oracle Human Resources は柔軟性があるので、企業におけるキャリア計画と後任計画は、いくつかあるモデルのいずれかを用いて処理できます。どのモデルを使用するかは、企業のキャリア計画と後任計画が役職と職階のどちらに基づいているか、インタフェースは Windows だけか、Web と Windows を併用しているかによって決まります。

キャリア・パスは、企業の規定に基づいた昇進・昇格のルートを示します。キャリア・パスは役職と職階の両方に設定できます。

役職や職階の後任を計画することで、該当する応募者のリストをいつでも用意できます。また、どのような研修や開発が必要かを識別し、特定の従業員を役職や職階に向けて準備し、さまざまな後任オプションを計画できます。

## 役職を基準したキャリア計画と後任計画のモデル

キャリア計画と後任計画が役職を基準にしている場合、ある役職から他の複数の役職への昇進・昇格の可能性を示すためにキャリア・パスを使用できます。

> **注意：**米国では、AAP- 要員分析レポートでキャリア・パス機能を使用して、AAP 計画に含まれる役職に対する支給ポイント行を作成します。

「キャリア・パス名」ウィンドウと「キャリア・パス・マップ」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「キャリア・パスの定義」

## ステップ 172キャリア・パスを作成してマップする

キャリア・パスは、雇用する個人でなく企業の構造を基準にしています。また、各従業員に対して個人的な目標や昇進・昇格パスを記録することが必要な場合もあります。これを実現するには複数の方法があります。

「キャリア・パス名」ウィンドウと「キャリア・パス・マップ」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「キャリア・パスの定義」

## ステップ 173勤務選択を入力する

個人のキャリア計画を指定しやすくするために、勤務選択を使用できます。

「勤務選択」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「役職または職階の勤務選択の入力」

## 職階を基準としたキャリア計画と後任計画のモデル

企業のキャリア計画と後任計画が職階を基準にしている場合、別の職階階層を作成して任意のタイプの昇進・昇格が表示できます。こうすることで、既存のライン管理構造を表示したり、部門や役職タイプも表示できます。

## ステップ 174職階階層を作成する

必要があれば、典型的な昇進・昇格を表示するために、キャリア・パスを示す職階階層を作成します。

「職階階層」ウィンドウを使用します。

> **関連項目：**『Oracle HRMS ユーザーズ・ガイド』の「職階階層の作成」

### ステップ 175後任計画を使用する（ライン・マネージャの職責で SSHR を使用する）

SSHR を使用する場合、後任計画機能を使用して各従業員の次の職階を 1 つ以上記録できます。また、使用可能な職階ができたとき、適合する従業員グループを作成し、照合できます。

SSHR の後任計画機能を使用します。

### ステップ 176適合性照合を使用する（ライン・マネージャの職責で SSHR を使用する）

SSHR を使用する場合、適合性照合機能を使用して従業員のコンピテンス・プロファイルを職階に必要なコンピテンスと照合できます。

SSHR の適合性照合機能を使用します。

### ステップ 177添付または特別情報タイプを使用する

特別情報タイプを使用するか、添付の形で後任計画を従業員に対して保持することを考えます。

「個人特別情報キー」フレックスフィールドを使用します。

**関連項目：**「特別情報タイプの定義」

# 第 II 部

## テクニカル・エッセイ

# 3

# デートトラック

## デートトラックの機能

デートトラックは、アプリケーションのデータベースに時系列にデータを追加します。デートトラックされるレコードの値は、データの表示開始日によって異なります。たとえば、1992 年 7 月 12 日の有効日で従業員の年俸を問合せた場合と、1992 年 12 月 1 日の有効日で従業員の年棒を問合せた場合とでは、デートトラックされるレコードの値は異なります。しかし、アプリケーションとユーザーは、その従業員の給与を単一のレコードと見なします。

## デートトラック・フォームの動作

この項では、デートトラック機能を取り込むフォームの動作について説明します。

デートトラック・フォームのレコードを更新または削除しようとすると、選択肢がいくつか表示され、入力を促されます。この項では、それらの選択肢とそのデートトラック表への影響について説明します。

「今日」とは、ユーザーが設定した有効日のことです。

### 更新

ユーザーが現在のコミット単位でデートトラック・ブロックのフィールドを初めて変更しようとすると、更新を促す次の 2 つの選択肢が表示されます。

- UPDATE（更新）- 更新済値が新規行としてデータベースに書き込まれます。これらの値は、今日から 4712 年 12 月 31 日まで有効です。前の値は、昨日までの値を含み有効です。
- CORRECTION（訂正）- 更新済値が前のレコード値を上書きし、同じ有効日を継承します。

ユーザーが「UPDATE」を選択すると、デートトラックによって更新されるレコードの開始日が今日かどうか確認されます。今日である場合、前の値が失われる可能性があることを警告するメッセージが表示されます（これは、デートトラックでは日単位での情報の格納しかできないためです）。さらに、そのレコードのモードが「CORRECTION」に変更されます。

さらに、「UPDATE」を選択すると、デートトラックによって更新されるレコードの先の更新が入力されているかどうか確認されます。そのレコードが先日付で更新されている場合には、さらに次のような更新タイプの入力を促されます。

- UPDATE_CHANGE_INSERT（挿入）- ユーザーが行った変更は、現在のレコードの有効終了日まで有効です。その時点で、先日付の変更予定が有効になります。
- UPDATE_OVERRIDE（置換）- ユーザーが行った変更は、今から先日付の最後のレコードの終了日まで有効です。先日付を持つ変更はすべて削除されます。

ほとんどのフォームでは、更新する各レコードの更新モードをユーザーが指定する必要があります。フォームによっては、最初のレコードを更新する時のみ更新モードの指定が要求されることもあります。その場合、更新される他の行もすべて、同じ更新モード下に置かれます。顕著な変更をコミットまたは消去するまでは、再度入力を促されることはありません。

## 削除

レコードを削除しようとすると、削除のタイプを入力するように促されます。次の 4 つのオプションが提供されています。

- DELETE（終了日）- デートトラック削除です。ユーザーが現在表示しているレコードの有効終了日が今日の日付に設定されます。レコードはフォームから削除されますが、ユーザーが再問合せすることは可能です。
- ZAP（パージ）- 全削除です。日付スタンプに関係なく、キー値と一致するすべてのレコードが削除されます。
- FUTURE CHANGE（全て）- このオプションを選択すると、先日付デートトラック削除など、現在のレコードに対する先日付変更がすべて削除されます。現在のレコードの有効終了日が 4712 年 12 月 31 日に設定されます。

  再問合せするとレコードが再表示されます。
- DELETE NEXT CHANGE（次の変更）- このオプションを選択すると、現在のデートトラック・レコードに対する次の変更が削除されます。

  このレコードに別の先日付のデートトラック行が存在している場合には、その行は削除され、現在の行の有効終了日が削除された行の有効終了日に設定されます。

  先日付のデートトラック行が 1 つもなくても、現在の行の終了日が 4712 年 12 月 31 日以外の日付に設定されている場合には、このオプションを選択すると、有効終了日が 4712 年 12 月 31 日に設定されます。このため、有効終了日の変更と見なされます。

  このオプションを選択した場合には、再問合せによって再表示できるものの、現在の行がフォームから再度削除されるということに注意してください。

### 挿入

レコードの挿入時にはユーザーがモードを指定する必要はありません。有効開始日は常に今日（有効日）に設定されます。有効終了日は可能なかぎり先の日付に設定されます。通常は4712 年 12 月 31 日になりますが、特にレコードに親デートトラック・レコードがある場合などには、これよりも前になることがあります。

## デートトラック表の表体系

デートトラック（DT）レコードは、キー値ごとに 1 つずつ生成されるレコードで、アプリケーションとユーザーの両者によって確認されます。この DT レコードは時間の経過に伴って変化する場合があるので、データベース内の 1 つ以上の物理行に対応することがあります。レコードの履歴は、レコードの作成時に行を格納し、レコードの変更時に常に追加行を格納することによって保持されます。これらの行を管理するために、すべてのデートトラック表に次の列を含める必要があります。

有効開始日（EFFECTIVE_START_DATE）NOT NULL の日付

有効終了日（EFFECTIVE_END_DATE）NOT NULL の日付

有効開始日は、レコードが挿入された日付を示します。有効終了日は、レコードが削除または更新された日付を示します。削除レコードはそのキーに関するすべての行の中で最も日付の大きい終了日を持ちますが、更新レコードの場合は、そのキーに関する少なくとも 1 つの行はさらに日付の大きい有効終了日を持ちます。

時間はサポートされていないので、有効開始日は 00 時 00 分に始まり、有効終了日は 23 時 59 分に終わります。このため、DT レコードが変更されるのは多くても 1 日に 1 回です。

### 例

**表 3-1 デートトラック表の内容の例**

| 従業員 ID (EMPID) | 従業員名 (EMPNAME) | 給与 (SALARY) | EFFECTIVE_ START_DATE | EFFECTIVE_ END_DATE |
|---|---|---|---|---|
| 3203 | SMITH | 17,000 | 12-MAR-1989 | 19-JUL-1989 |
| 3203 | SMITH | 18,200 | 20-JUL-1989 | 20-JUL-1989 |
| 3203 | SMITH | 18,400 | 21-JUL-1989 | 01-DEC-1989 |

この表は、ユーザーが次の設定を行った後の物理表を示したものです。

- 有効日を 1989 年 3 月 12 日に設定。SMITH に関するレコードを挿入。
- 有効日を 1989 年 7 月 20 日に設定。SMITH に関するレコードを新しい給与に更新。
- 有効日を 1989 年 7 月 21 日に設定。SMITH に関するレコードを新しい給与に再度更新。
- 有効日を 1989 年 12 月 1 日に設定。SMITH に関するレコードを削除。

次の表は、SMITH に関するレコードを様々な有効日に問合せた場合の表示結果を示したものです。

**表 3-2 デートトラック表の問合せ結果の例**

| 有効日 (EFFECTIVE DATE) | 従業員 ID (EMPID) | 従業員名 (EMPNAME) | 給与 (SALARY) |
|---|---|---|---|
| 11-MAR-1989 | ** 取出し行なし | | |
| 12-JUN-1989 | 3203 | SMITH | 17,000 |
| 21-JUL-1989 | 3203 | SMITH | 18,400 |
| 02-DEC-1989 | ** 取出し行なし | | |

表内の主キー列はもはや一意ではないため、主キー列が含まれていた表の索引には現在、有効開始日（EFFECTIVE_START_DATE）と有効終了日（EFFECTIVE_END_DATE）の 2 つの列も含まれているはずです。

### デートトラック表のリスト

Oracle Human Resources で使用されるデートトラック表のリストを入手するには、データ・ディクショナリから Application Short Name%F などの表名を選択してください。目的の人事管理システム・アプリケーションの短いコードで置き換えます（PER または BEN など）。

各デートトラック表には、<TABLE NAME> と呼ばれるデートトラック・ビューと、<TABLE NAME_F> と呼ばれる全表を指すシノニムがあります。

## デートトラック表とビューの作成

前項では、デートトラック表の表体系について説明しました。この項では、デートトラック表とビューの作成手順について説明します。

次のデートトラック表命名体系を使用してください。

実表 : <TABLE NAME_F>

デートトラック・ビュー : <TABLE NAME>

デートトラック・ビューの他にも、SYSDATE 時点の表内の行を表すビューがあります。このビューの名称は、表名の終わりにある _F を _X に置き換えることによって導出されます。

### 例

EMPLOYEES という既存の表にデートトラックを組み込むには、次の手順を実行します。

1. EMPLOYEES と同じではあるものの、有効開始日（EFFECTIVE_START_DATE）と有効終了日（EFFECTIVE_END_DATE）の 2 つの列が追加された EMPLOYEES_F という新しい表を作成します。通常は、これらの 2 つの列は最大範囲に設定します。

```
CREATE TABLE EMPLOYEES_F AS
SELECT EMPLOYEES.*,
     TO_DATE('01-01-0001', 'DD-MON-YYYY') EFFECTIVE_START_DATE,
     TO_DATE('31-12-4712','DD-MON-YYYY') EFFECTIVE_END_DATE
FROM EMPLOYEES;
ALTER TABLE EMPLOYEES_F
MODIFY (EFFECTIVE_START_DATE NOT NULL,
      EFFECTIVE_END_DATE NOT NULL);
```

前の表を削除します。

```
DROP TABLE EMPLOYEES
```

前の表にこれら 2 つの新しい列がすでにある場合には、改名のみしてください。

```
RENAME EMPLOYEES TO EMPLOYEES_F;
```

2. 前の索引を削除し、新しい一意索引を前の索引＋有効開始日（EFFECTIVE_START_DATE）＋有効終了日（EFFECTIVE_END_DATE）の形で作成し、さらに新しい非一意索引を前の非一意索引と同じ形で作成することによって、デートトラック表の新しい一意索引を作成します。
3. EMPLOYEES というデートトラック・ビューを作成します。このビューには、有効日の現ユーザー有効 ID の FND_SESSIONS に対する入力が使用されます。

```
CREATE VIEW EMPLOYEES AS
SELECT *
FROM   EMPLOYEES_F
WHERE  EFFECTIVE_START_DATE <=
  (SELECT EFFECTIVE_DATE
  FROM    FND_SESSIONS
  WHERE   FND_SESSIONS.SESSION_ID = USERENV('SESSIONID'))
AND    EFFECTIVE_END_DATE >=
(SELECT EFFECTIVE_DATE
  FROM   FND_SESSIONS
  WHERE  FND_SESSIONS.SESSION_ID = USERENV('SESSIONID'))
```

4. EMPLOYEES_F 表を基に EMPLOYEES_X ビューを作成するには、次の SQL を使用します。

```
CREATE VIEW EMPLOYEES_X AS
SELECT *
FROM   EMPLOYEES_F
WHERE  EFFECTIVE_START_DATE <= SYSDATE
AND    EFFECTIVE_END_DATE >= SYSDATE
```

# フォーム・ユーザーが使用できるデートトラック・オプションの制限

ユーザーがデートトラック・レコードを編集または削除すると、どのようなタイプの更新または削除を実行するかをユーザーに確認するウィンドウが表示されます。システムはこのウィンドウを表示する前に、カスタム・ライブラリ・イベント（DT_SELECT_MODE と呼ばれる）をコールします。このイベントからは、デートトラックに通常表示されるボタンのリスト（更新や訂正など）が渡されます。

ユーザーのカスタム・コードによって、表示されるボタンを制限できます。必要であれば、ユーザーが更新オプションや削除オプションを使用できないようにして、エラー・メッセージを表示するようにできます。しかし、有効日やユーザーが実行している操作など、デートトラックが通常はそのエンティティで表示しないボタンを表示することはできません。

ユーザーが「更新」を選択した場合、先日付の変更がある場合には、このユーザーに「挿入」および「置換」という 2 つの更新オプションを提示するかどうかをカスタム・コードで決定できるように、カスタム・ライブラリ・イベント・ポイントが再度実行されることがあります。

## グローバル変数

DT_SELECT_MODE イベントでは次のグローバル変数を使用できます。これらのグローバル変数は他のカスタム・ライブラリ・イベントでは使用できません。

**表 3-3 DT_SELECT_MODE イベントのグローバル変数**

| グローバル変数名 | 読込み / 書込み | 説明 |
|---|---|---|
| g_dt_update | 読込みおよび書込み | モード選択ウィンドウに通常「更新」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_correction | 読込みおよび書込み | モード選択ウィンドウに通常「訂正」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_update_change_insert | 読込みおよび書込み | モード選択ウィンドウに通常「挿入」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |

**表 3-3 DT_SELECT_MODE イベントのグローバル変数（続き）**

| グローバル変数名 | 読込み / 書込み | 説明 |
|---|---|---|
| g_dt_update_override | 読込みおよび書込み | モード選択ウィンドウに通常「置換」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_zap | 読込みおよび書込み | モード選択ウィンドウに通常「パージ」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_delete | 読込みおよび書込み | モード選択ウィンドウに通常「終了日」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_future_change | 読込みおよび書込み | モード選択ウィンドウに通常「全て」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |
| g_dt_delete_next_change | 読込みおよび書込み | モード選択ウィンドウに通常「次」ボタンが表示される場合は TRUE に設定される。その他の場合は FALSE に設定される。 |

**注意：** カスタム・コードを使用して TRUE 値を FALSE に変更できます。しかし、FALSE 値を TRUE に変更しようとしても、システムはこの変更を無視します。

## DT_SELECT_MODE イベントの有効化

DT_SELECT_MODE イベントを有効にするには、CUSTOM パッケージ、CUSTOM ライブラリの中の STYLE プロシージャに次のコードを追加します。

```
if event_name = 'DT_SELECT_MODE' then
  return custom.after;
else
  return custom.standard;
end if;
```

## カスタム・コードの例

デートトラックで通常は「モード選択」ウィンドウの「削除」モード・ボタンが使用可能になっている状況で、このボタンを表示しないことにするとします。CUSTOM パッケージ、CUSTOM ライブラリの中の EVENT プロシージャに次のコードを追加します。

```
if (event_name = 'DT_SELECT_MODE') then
  if name_in('GLOBAL.G_DT_DELETE') = 'TRUE' then
    copy('FALSE', 'GLOBAL.G_DT_DELETE');
  end if;
end if;
```

# デートトラック履歴ビューの作成および修正

デートトラック履歴は、デートトラック情報を入力できるウィンドウであれば大抵表示できます。デートトラック履歴を使用すると、レコードおよびフィールドに加えられた変更、およびその変更を誰が行ったかを把握できます。確認するフィールドを選択することによって、選択したフィールドの時間と伴に変化した値を表示できます。

デートトラック履歴を表示するには、ツールバー上のボタンを選択します。

## 作成および変更できるものは？

新しいビューを作成することも、既存のビューを修正して情報の表示をカスタマイズすることも可能です。次のことを行えます。

- 他の表と結合するビューの作成。このため、有意な表名を列のヘッダーとして使用できます。これに対して、実表には別の表の ID しか表示できません。
- ビューの修正による、表示するフィールドの決定。
- ビューの修正による、付加フレックスフィールド・セグメントに対して定義した有意名に対する列名の別名の表示。
- ビジネス・グループ ID など、ユーザーの選択基準による、使用するビューの決定。

## デートトラック履歴を要求した場合に発生する処理

デートトラック履歴が要求されると、オラクル人事管理システムは 3 つのソースのうちの 1 つから情報を抽出します。アプリケーションは最初に、カスタム・ライブラリで指定された代替ビューを検索し、そのビューが存在する場合はそこから情報を抽出します。代替ビューが指定されていない場合は、次にデフォルト・デートトラック履歴ビューを検索して情報を抽出します。デフォルト・デートトラック履歴ビューが存在しない場合は、実表から情報を抽出します。その後、「デートトラック履歴変更フィールド要約」ウィンドウに情報を表示します。

デフォルト・デートトラック履歴ビューの名称は、接尾辞 _F が _D に置き換わる以外は、対応する実表と同じです。たとえば、実表が PER_ALL_PEOPLE_F である場合には、アプリケーションは PER_ALL_PEOPLE_D というビューを検索します。

**注意：** 1 つのデートトラック表に対して複数の履歴ビューを定義することができます。このため、履歴ビューの名称がこの命名規則どおりになっていないこともあります。

ビューが存在する場合、アプリケーションはエンティティ名および列プロンプトに関する情報を、次のデートトラック表から読み取ります。

- DT_TITLE_PROMPTS_TL
- DT_DATE_PROMPTS_TL
- DT_COLUMN_PROMPTS_TL

DT_COLUMN_PROMPTS_TL 表から列情報を入手できない場合は、ビュー定義から情報が入手されます。デートトラック履歴コードは、表またはビューの列名を変更した後で表示します。アンダースコアはスペースに置き換えられ、各ワードの最初の文字は大文字で表示されます。

## デートトラック履歴ビューの修正 / 変更に関するルール

デートトラック履歴ビューの名称は、接尾辞 _F が _D に置き換わる以外は、可能な限り対応する実表と同じ名称にする必要があります。カスタム・ライブラリを使用して代替ビューを指定した場合は、ビュー名が変わりますが、それでも _D 接尾辞は必要です。

すべてのビューに次の列が含まれている必要があります。

- 実表の主キー
- 実表の有効開始日
- 実表の有効終了日
- 最終更新日列
- 最終更新者列（FND_USER_VIEW への外部結合によって、実際のユーザー名を入手します）。

> **注意：** デートトラック履歴ビューの列の数は 35 に制限されています。ビューには、主キー、有効開始日、有効終了日の各列を含める必要がありますが、それらの列を「デートトラック履歴」ウィンドウに表示することはできません。

システム定義のデートトラック・ビュー作成スクリプトは絶対に編集しないでください。システム定義のデートトラック履歴ビューをカスタマイズする場合は、スクリプトのコピーを作成し、コピーに修正を加えてください。アップグレード後は、カスタマイズした内容がアップグレード版に定義されている新しいビューと一貫性があるかどうかは、お客様がご確認ください。一貫性がある場合には、カスタマイズしたビュー作成スクリプトを再実行して、カスタマイズしたビューを再作成することができます。

## 列を追加する場合のフォルダ定義の更新

デートトラック履歴ビューに列を追加すると、列順序に影響を与えることがあります。また、以前にフォルダを保存してある場合は、表示されるデータおよびプロンプトが一致しなくなる可能性があります。これは、「デートトラック履歴変更フィールド要約」ウィンドウには列名がアルファベット順に表示されますが、最初の 2 つの列には有効日値が表示されるためです。

データベースに新規ビューを適用した直後に、すべてのフォルダ定義を更新することをお薦めします。そうしないと、その後に表示されるデータおよびフォルダ内のプロンプトが一致しなくなる可能性があります。

## デートトラック履歴ビューの例

この例の場合、実表は PAY_GRADE_RULES_F です。

```
     create or replace view pay_grade_rules_d
     (grade_rule_id,
     effective_start_date,
     effective_end_date,
     maximum,
     mid_value,
     minimum,
     grade,
     rate_type,
     last_update_date,
     last_updated_by)
    AS
    select GRULE.grade_rule_id,
           GRULE.effective_start_date,
           GRULE.effective_end_date,
           GRULE.maximum,
           GRULE.mid_value,
           GRULE.minimum,
           GRADE.name,
           HR1.meaning,
           GRULE.last_update_date,
           FUSER.user_name
    from   pay_grade_rules_f          GRULE
    ,      per_grades                 GRADE
    ,      hr_lookups                 HR1
    ,      fnd_user_view              FUSER
    where  GRADE.grade_id           = GRULE.grade_or_spinal_point_id
and HR1.lookup_code              (+)= GRULE.rate_type
and HR1.lookup_type              (+)= 'RATE_TYPE'
and FUSER.user_id                (+)= GRULE.last_updated_by
```

## 代替デートトラック履歴ビューの使用方法

「デートトラック履歴変更フィールド要約」ウィンドウを表示する前に、システムはカスタム・ライブラリ・イベント（DT_CALL_HISTORY と呼ばれる）をコールします。これにより、現在のレコードの詳細と、製品で通常どのデートトラック・ビューが使用されるかの詳細が渡されます。カスタム・コードを書けば、デートトラック履歴で使用するビューの名称を変更することができます。コードに IF 文を含めることにより、様々な状況でどのビューを使用するかを決定できます。

**注意：** データベース内に代替ビューが存在し、関係するユーザーがビューへの選択アクセスを行えるようにすることは、ユーザーの責任です。

追加する各ビューについて、ビュー名に基づき、DT_TITLE_PROMPTS_TL 表および DT_COLUMN_PROMPTS_TL 表に追加の行を挿入する必要があります。SQL*Plus スクリプトを使用して、追加の表内容およびビュー定義の保守を行います。

### グローバル変数

DT_CALL_HISTORY イベントでは次のグローバル変数を使用できます。これらのグローバル変数は他の CUSTOM ライブラリ・イベントでは使用できません。

| グローバル変数名 | 読込み / 書込み | 説明 |
|---|---|---|
| g_dt_basetable | 読込み専用 | データが保持されるデータベース表の名称。例 :PER_ALL_PEOPLE_F |
| g_dt_uidfield | 読込み専用 | データベース表の代理 ID の名称。例 :PERSON_ID |
| g_dt_uidvalue | 読込み専用 | 現在のレコードの代理 ID 値。 |
| g_dt_alternative_history_view | 読込みおよび書込み | 通常デートトラック履歴は、_F 接尾辞が _D に変わることを除き、データベース表と同じ名称のデータベース・ビューで履歴データを問い合せます。この場合、このグローバル変数は NULL になります。たとえば、データベース表が PER_ALL_PEOPLE_F の場合には、PER_ALL_PEOPLE_D ビューが使用されます。別のビューを使用する場合は、このグローバル変数を実際のビュー名に変更します（変数が最初は NULL であっても）。 |

### DT_CALL_HISTORY イベントの有効化

DT_CALL_HISTORY イベントを有効にするには、CUSTOM パッケージ、CUSTOM ライブラリの中の STYLE プロシージャに次のコードを追加します。

```
if event_name = 'DT_CALL_HISTORY' then
  return custom.after;
else
  return custom.standard;
end if;
```

### カスタム・コードの例

標準製品で PER_ALL_PEOPLE_D ビューが使用される状況で、必ず別のビューを使用することにしたとします。CUSTOM パッケージ、CUSTOM ライブラリの中の EVENT プロシージャに次のコードを追加します。

```
if (event_name = 'DT_CALL_HISTORY') then
  if name_in('global.g_dt_basetable') = 'PER_ALL_PEOPLE_F' then
    copy
     ('NAME_OF_OTHER_VIEW'
     ,'global.g_dt_alternative_history_view'
     );
  end if;
end if;
```

## デートトラック履歴ビューのリスト

システム定義のビューおよびビュー作成スクリプトを次に示します。

**表 3-4 デートトラック履歴ビュー**

| ビュー名 | ベースの表 | ビュー作成スクリプト |
|---|---|---|
| BEN_BENEFIT_CONTRIBUTIONS_D | BEN_BENEFIT_CONTRIBUTIONS_F | pedtbbcf.sql |
| HXT_ADD_ASSIGN_INFO_D | HXT_ADD_ASSIGN_INFO_F | hxtdtaas.sql |
| HXT_ADD_ELEM_INFO_D | HXT_ADD_ELEM_INFO_F | hxtdtael.sql |
| HXT_SUM_HOURS_WORKED_D | HXT_SUM_HOURS_WORKED_F | hxtdtsum.sql |
| HXT_TIMECARDS_D | HXT_TIMECARDS_F | hxtdttim.sql |
| PAY_ALL_PAYROLLS_D | PAY_ALL_PAYROLLS_F | pydtpayr.sql |
| PAY_BALANCE_FEEDS_D | PAY_BALANCE_FEEDS_F | pydtbalf.sql |
| PAY_CA_EMP_FED_TAX_INFO_D | PAY_CA_EMP_FED_TAX_INFO_F | pycadtfd.sql |
| PAY_CA_EMP_PROV_TAX_INFO_D | PAY_CA_EMP_PROV_TAX_INFO_F | pycadtpv.sql |
| PAY_COST_ALLOCATIONS_D | PAY_COST_ALLOCATIONS_F | pydtpcst.sql |
| PAY_ELEMENT_LINKS_D | PAY_ELEMENT_LINKS_F | pydtelin.sql |

**表 3-4 デートトラック履歴ビュー（続き）**

| ビュー名 | ベースの表 | ビュー作成スクリプト |
|---|---|---|
| PAY_ELEMENT_TYPES_D | PAY_ELEMENT_TYPES_F | pydtetyp.sql |
| PAY_FORMULA_RESULT_RULES_D | PAY_FORMULA_RESULT_RULES_F | pydtfmrr.sql |
| PAY_GRADE_RULES_D | PAY_GRADE_RULES_F | pydtgrdt.sql |
| PAY_INPUT_VALUES_D | PAY_INPUT_VALUES_F | pydtinpv.sql |
| PAY_LINK_INPUT_VALUES_D | PAY_LINK_INPUT_VALIES_F | pydtliiv.sql |
| PAY_ORG_PAYMENT_METHODS_D | PAY_ORG_PAYMENT_METHODS_F | pydtpaym.sql |
| PAY_PERSONAL_PAYMENT_METHODS_D | PAY_PERSONAL_PAYMENT_METHODS_F | pydtppym.sql |
| PAY_STATUS_PROCESSING_RULES_D | PAY_STATUS_PROCESSING_RULES_F | pydtstpr.sql |
| PAY_USER_COLUMN_INSTANCES_D | PAY_USER_COLUMN_INSTANCES_F | pydtucin.sql |
| PAY_USER_ROWS_D | PAY_USER_ROWS_F | pydtussrr.sql |
| PER_ALL_ASSIGNMENTS_D | PER_ALL_ASSIGNMENTS_F | pedtasgn.sql |
| PER_ALL_PEOPLE_D | PER_ALL_PEOPLE_F | pedtpepl.sq |
| PER_ASSIGNMENT_BUDGET_VALUES_D | PER_ASSIGNMENT_BUDGET_VALUES_F | pedtabv.sql |
| PER_COBRA_COVERAGE_BENEFITS_D | PER_COBRA_COVERAGE_BENEFITS_F | pedtccbf.sql |
| PER_GRADE_SPINES_D | PER_GRADE_SPINES_F | pedtgrsp.sql |
| PER_SPINAL_POINT_PLACEMENTS_D | PER_SPINAL_POINT_PLACEMENTS_F | pedtsppp.sql |
| PER_SPINAL_POINT_STEPS_D | PER_SPINAL_POINT_STEPS_F | pedtspst.sql |
| PER_PERSON_TYPE_USAGES_D | PER_PERSON_TYPE_USAGES_F | pedtptu.sql |
| PER_CONTRACTS_D | PER_CONTRACTS_F | pedtctc.sql |

# 4

# バッチ・エレメント・エントリ

## バッチ・エレメント・エントリのバッチ合計制御の作成処理

バッチ合計制御は、特定のユーザー要件にあわせてバッチ・コンテンツの検証をカスタマイズするための方法を提供します。こうした検証は、たとえば、バッチ・ラインで「合計」または「平均」処理を実行し、その結果値をユーザーが入力した値と照合することによって行われます。

バッチは、「バッチ・ヘッダー」ウィンドウ、およびこのウィンドウから表示可能なその他のウィンドウから入力および表示できます。

### 合計制御の設定

合計制御タイプは、バッチ内のライン数のチェックのために事前定義されています（制御タイプ = ライン合計）。

合計制御を作成すると、参照タイプ CONTROL_TYPE への参照を定義することで、数値エレメント入力パラメータの合計を計算できます。

その他の種類の合計制御が必要な場合は、その合計制御への参照を定義できますが、バッチで合計をチェックするための検証プロシージャを作成する必要があります。次のセクションでは、この検証プロシージャの作成方法について説明します。

### SQL コードの作成

次のプロシージャは、NULL 文を含んだ状態で提供されます。ユーザーのカスタマイズした合計制御検証コードで NULL 文を置き換えてください。

- プロシージャ : check_control
- パッケージ : user_check
- ファイル : pyusrchk.pkb

## パラメータ

check_control プロシージャは、BEE プロセスのバッチ検証段階で実行されます。このプロシージャに渡されるパラメータは次のとおりです。

- p_batch_id　　バッチ ID です。
- p_control_type　　合計制御の名前です。
- p_control_total　　比較のためにユーザーが入力した値です。

このプロシージャでは、この他にも 2 つのパラメータ（p_status と p_message）が、バッチ合計制御検証が失敗に終わった場合にシステムにエラー・コードとメッセージを返すのに用いられます。

## バッチ・ライン

バッチ・データの各ラインは、pay_batch_lines 表にレコードとして格納されます。データは、value_1 から value_15 までのフィールドに格納されます。フィールドの番号は、「バッチ・ライン」ウィンドウ内の列に対応します。

たとえば、チェック桁を検証する場合、次の PL/SQL コードを基本として使用できます。

```
PROCEDURE check_control
  (
    p_batch_id          IN      NUMBER,
    p_control_type      IN      VARCHAR2,
    p_control_total     IN      VARCHAR2,
    p_status            IN OUT  VARCHAR2,
    p_message           OUT     VARCHAR2
  ) IS
  total NUMBER;
BEGIN
-- Check the control type is the one we're expecting
IF p_control_type = 'CHECK_DIGIT' THEN
-- Calculate the MOD 10 of total values in value_1
    SELECT MOD(NVL(SUM(value_1),0),10) INTO total FROM pay_batch_lines
    WHERE batch_id = p_batch_id;
-- Compare with the user entered value
    IF total <> p_control_total THEN
-- Create the error message to return and set the status to E(rror)
      p_message := 'Control total TOT1 (' || p_control_total ||
                   'does not match calculated value (' || total ||
                   ')';
      p_status := 'E';
    ENDIF;
  ENDIF;
END check_control;
```

しかし、これは非常に簡単な例です。同じバッチ内のバッチ・ラインが 1 つ以上のエレメントに対して入力された場合には、値列がエレメントによって異なる可能性があります。入力パラメータ 'Identification' のチェック桁を検証するための、より複雑な例を次に示します。

```
PROCEDURE check_control
  (
    p_batch_id          IN      NUMBER,
    p_control_type      IN      VARCHAR2,
    p_control_total     IN      VARCHAR2,
    p_status            IN OUT  VARCHAR2,
    p_message           OUT     VARCHAR2
  ) IS
    CURSOR c1 IS
      SELECT DISTINCT element_type_id
        FROM pay_batch_lines
        WHERE batch_id = p_batch_id;
--
    r1 c1%ROWTYPE;
    total NUMBER;
    value_num NUMBER;
    sqlstr VARCHAR2(200);
    c2 INTEGER;
    ret INTEGER;
  BEGIN
--
-- Check the control type is the one we're expecting
    IF p_control_type = 'CHECK_DIGIT2' THEN
      total := 0;
--
-- Loop through each element in the batch lines
      FOR r1 IN c1 LOOP
--
-- Find out the value number that 'Identification' is in
        SELECT display_sequence
          INTO value_num
          FROM pay_input_values iv,
               pay_batch_headers bh,
               pay_element_types et
          WHERE bh.batch_id = p_batch_id AND
                iv.business_group_id = bh.business_group_id AND
                et.element_type_id = r1.element_type_id AND
                iv.element_type_id = et.element_type_id AND
                iv.name = 'Identification';
--
-- Create an SQL string to add the values
        sqlstr := 'SELECT MOD(NVL(SUM(value_' || value_num ||
                  '),0),10) ' ||
```

```
                    'FROM pay_batch_lines ' ||
                    'WHERE batch_id = ' || p_batch_id || ' AND '
                     ||'element_type_id = ''' ||
                     r1.element_type_id || '''';
--
-- Call the string using dynamic SQL and put the value in 'total'
        c2 := dbms_sql.open_cursor;
        dbms_sql.parse (c2,sqlstr,dbms_sql.v7);
        dbms_sql.define_column (c2,1,total);
        ret := dbms_sql.execute (c2);
        ret := dbms_sql.fetch_rows (c2);
--
-- Check we got some values back
        if ret > 0 then
          dbms_sql.column_value (c2,1,total);
        else
          total := 0;
        end if;
--
        dbms_sql.close_cursor (c2);
--
-- Check the total matches the user entered value and create an
-- error message if it doesn't
         IF total <> p_control_total THEN
          p_message := 'Check digit expected '||p_control_total||
          ' but got '||to_char(total);
          p_status := 'E';
         END IF;
      END LOOP;
    END IF;
  END check_control;
```

# 5

# 給与処理

## 概要

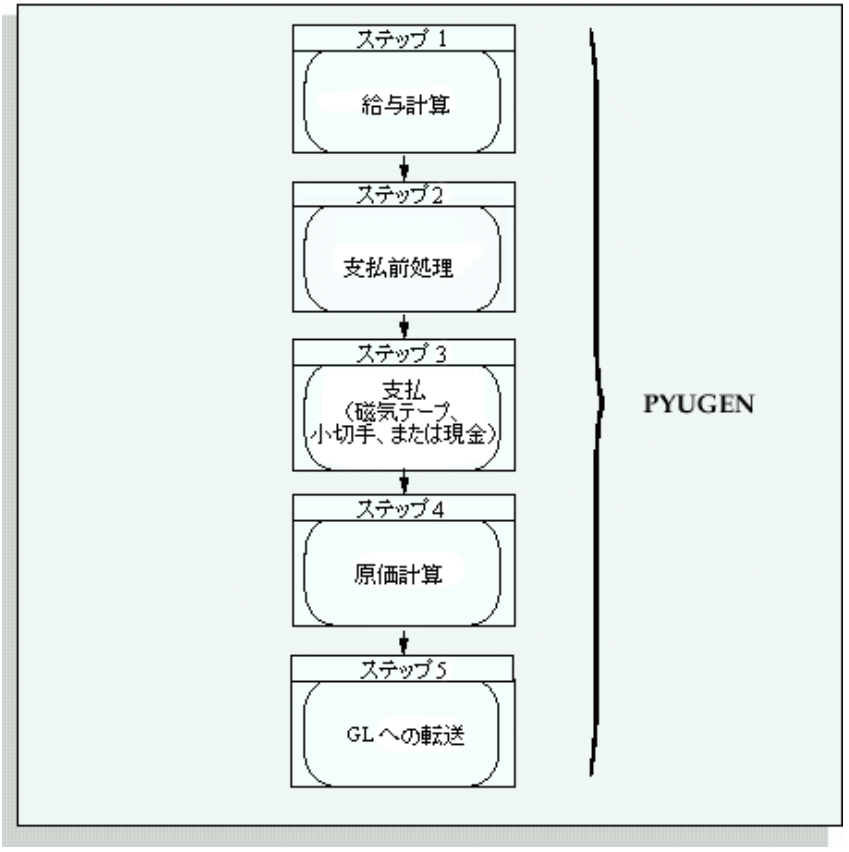
Oracle Payroll は、ビジネス・ニーズに合った最適な方法で通常の支払サイクルを実行するためには欠かせない、優れた柔軟性を備えています。オラクル社では、これを実現するために、PYUGEN と呼ばれるモジュール方式のバッチ処理を提供しています。

## PYUGEN

PYUGEN は、複数の処理を実行できる汎用プロセスです。Oracle Payroll のシステム管理者は、特定のパラメータ・セットやデフォルトを使用して登録することにより、実行できる処理を指定します。

パラメータは、実行する特定の給与処理を指定します。これらは Oracle Payroll に事前定義されています。値はユーザーには見えません。

次の図は、PYUGEN によって実行される給与処理と、それらの処理が実行される際の典型的な順序を示したものです。個々の処理は、従業員への支払を計算し、発生させるのに必要な各種処理を実行します。




## 登録詳細のチェック

「コンカレント・プログラム」ウィンドウを使用して、各給与処理の登録詳細をチェックすることができます。これらの詳細は事前定義されており、変更は禁止されています。インプリメンテーション時には、独自のバージョンの給与処理を追加して、ユーザーの支払サイクルの実行を簡略化することもできます。たとえば、必要に応じて、給与ごとに次の異なる給与計算処理を定義することもできます。

- 名称
- セキュリティ
- ユーザーごとに異なるデフォルト値

コンカレント・プログラムの登録の詳細は『Oracle Applications システム管理者ガイド』を参照してください。

## 給与計算処理パラメータ

給与計算処理パラメータは、Oracle Payroll バッチ処理の様々な局面を制御するシステム・レベルのパラメータです。特定のパラメータに対して値を設定した場合には、システム全体にその影響が及ぶ可能性があるということを認識しておくことが大切です。

参照：給与計算処理パラメータ：5-50 ページ

## 給与処理の概要

支払サイクルで最初に実行する処理は、給与計算処理です。この処理では、従業員に対する総支給額と差引支給額が計算されます。給与計算が完了したら、支払前処理を実行します。この処理では、従業員が要求した支払方法別に従業員の支払が分配されます。さらに、支払が第三者に割り当てられます。

次に、下記の支払処理のうち 1 つを開始して、従業員に対する支払を発生させます。

- MAGTAPE（たとえば、UK の場合の BACS または US の場合の NACHA）
- CHEQUE（小切手ライター）- JP 以外
- CASH（現金）- UK、JP のみ

これらの支払処理では、各支払タイプに割り当てられた未払い支払前処理値が呼び出され、必要な支払ファイルが作成されます。従業員に対する支払を実際に発生させるのは、これらの処理です。

原価計算処理では、給与計算結果が原価セグメントに割り当てられます。GL への転送処理では、原価情報が Oracle General Ledger インタフェース表に転送されます。

**関連項目**

給与計算処理：5-5 ページ

支払前処理：5-16 ページ

支払処理：5-21 ページ

- 磁気テープ処理：5-21 ページ
- 小切手ライター処理（JP 以外）：5-41 ページ
- 現金処理：5-49 ページ

原価計算処理：5-67 ページ

GL への転送処理：5-66 ページ

### サポート処理

このような通常のアクティビティ・サイクル以外にも、各サイクルの訂正や完了をサポートする処理があります。次のものが含まれます。

- 再実行にマーク付け
- 再実行
- ロールバック
- QuickPay
- 遡及計算
- 前払（JP 以外）
- アーカイブ

これらサポート処理の詳細は『Oracle HRMS ユーザーズ・ガイド』を参照してください。アーカイブ処理については、6-1 ページの「給与アーカイブ・レポータ（PAR）処理」を参照してください。

## アサイメント・レベル連動

PYUGEN の支払計算順序は、処理が正常に実行されるために非常に重要です。これは、各処理が前の処理の結果を使用し、それに基づいて行われるためです。処理する順序もまた、データの整合性の問題により決定されます。たとえば、支払前処理（支払方法に応じて支払を準備する）では給与計算処理（総支給額と差引支給額を計算する）の結果が使用されます。

支払を正しく行うためには、支払前の結果も変更しないかぎり、給与計算処理の結果を変更することはできません。Oracle Payroll はアサイメント・レベル連動ルールを使用してこれを行います。

参照：アサイメント・レベル連動：5-61 ページ

# 給与計算処理

「給与計算」処理では、従業員に対する総支給額と差引支給額が計算されます。

この処理では、給与計算処理を使用して各給与計算が表されます。また、どのアサイメントで給与処理計算が実行されているかが識別されます。その処理が、給与タイプのアサイメント処理となります。

アサイメントの各エレメントの処理結果が実行結果値になります。これらの個々の結果は、総支給額と差引支給額が要約されたバランス、特に、支払バランスに累積されます。支払バランスは、通常の支払サイクルにおいて次の処理となる支払前処理に引き継がれます。

## アサイメントとエレメントの決定

「給与計算」処理の最初の段階は、現在のバッチに含めるアサイメントとエレメントを決定することです。ユーザーは処理を開始するときに、アサイメント・セットとエレメント・セットを選択することにより、これらを指定します。デフォルトは「すべて」です。

「給与計算」は、いくつかの特定のエンティティにアクセスして処理を行います。また、それらのエンティティが選択、更新、削除、挿入のいずれに使用されるのかを識別します。エンティティがデートトラックされる場合には、「給与計算」は変更されたデートトラック情報も識別し、それに応じて処理を行います。たとえば、デートトラックされるエンティティの更新には、表への実際の挿入が必要な場合もあります。

次のリストは、処理用のメイン・エンティティを示したものです。

キー : S ＝選択、U ＝更新、D ＝削除、I ＝挿入

| エンティティ名 | デートトラックか | 処理 |
|---|---|---|
| 給与処理 | No | S、U、I |
| アサイメント処理 | No | S、U、I |
| エレメント・エントリ | Yes | S、U |
| エレメント・エントリ値 | Yes | S、U |
| 個人最新バランス | No | S、U、I |
| アサイメント最新バランス | No | S、U、I |
| バランス・コンテキスト | No | S、U、I |
| 処理コンテキスト | No | S、I |
| 実行結果 | No | S、U、I |
| 実行結果値 | No | S、U、I |

## 各アサイメントの処理

「給与計算」は、各アサイメントに合った適切な処理を適用します。特定の給与処理に関しては、これはアサイメント処理によって識別されます。次の「疑似コード」は、発生する処理を表しています。

```
get assignment status();
if assignment status is 'Process' then
      load element entries and values ();
      load latest balances ();
      while(entries to process)
         create run results if necessary ();
         set up User Defined Context Area ();
         /* third party hook */
         get processing mode for entry ();
         if(we are not skipping) then
            look for formula to run ();
            if(there is formula to execute) then
               execute formula ();
               if(error detected) then
                  handle error ();
               end if
            end if
            post run results and feed balances ();
         end if
      end while
      flush run results and values ();
      write / update latest balances ();
end if
```

## エレメント・エントリ処理

エレメント・エントリは、総支給額と差引支給額計算に入力された入力値を保持しています。各入力値の処理結果は、実行結果値となります。各アサイメントを処理する前に、「給与計算」はそのアサイメントのエントリをすべてメモリーにロードします。これには、すべての挿入前実行結果と値が含まれます。

デフォルトでは、変動処理エントリは、現給与期間に処理されなかった場合にのみ取り出されます。固定処理エントリは、給与計算を発行すると必ず取り出され、処理されます。処理頻度、エレメント・スキップ Formula、またはエレメント・セットを使用して、固定処理エントリの挿入を制限してください。

固定処理エレメントに追加エントリがあった場合、「給与計算」では追加分は変動処理エントリとして処理されます。(US 版の Oracle Payroll では追加エントリは使用されません。)

## 処理優先順位

各アサイメントのエントリの処理順序は、エレメントの処理優先順位、さらには各エントリの下位優先順位によって決定します。下位優先順位が NULL の場合は、エントリは次によって順序付けられます。

1. 処理優先順位
2. element_type_id
3. 入力タイプ

「給与処理」は、US 以外の国別仕様の標準エントリと追加エントリを計算する前に、「上書き」と「置換」の 2 種類のエントリをチェックします。

下位優先順位が指定された場合には、それを反映してメモリー内のリストで順序付けが変更されます。「調整」エントリとターゲット・エントリは一緒にされます。

## 退職処理

「給与計算」は、退職アサイメントに対しては、入力処理ルールをインプリメントします。

US 仕様では、これは「給与処理」の対象日が退職の実日付とアサイメントの最終処理日の間にある場合に、そのアサイメントに対する未処理の変動処理エントリが存在しているときのみそのアサイメントが処理されるということを意味します。

US 以外の仕様の場合は、最終給与処理日を入力することも可能です。つまり、「給与処理」の対象日が最終給与処理日とアサイメントの最終処理日の間にある場合には、そのアサイメントに対する未処理の変動処理エントリが存在している場合にのみそのアサイメントは処理されます。

追加エントリは、退職のための変動処理と見なされます。

## 実行結果と値の作成

処理されるすべてのエントリに関して、実行結果が存在している必要があります。また、各入力値に関しては、実行結果値が存在する必要があります。挿入前までにすでに存在していない場合には、適切な実行結果および値がメモリー内に作成され、データベースに挿入されるため、いつでも「給与計算」処理を開始できます。

たとえば、「結果値」を入力した場合には、変動処理エントリが挿入前実行結果および値を持つ可能性があります。

挿入前値はロールバックまたは再実行操作のマークによって自動的に削除されるため、「給与計算」はそれらの値を再設定します。しかし、リバーサルのロールバックの場合には、事前挿入の変動処理値が再設定されます。

同時に、「給与計算」は給与に現在の換算レートを適用して、通貨変換を実行します。これは、エレメントの入力通貨コードと出力通貨コードが異なる場合に発生します。どのような入力通貨を持つエレメントでも定義できます。

エレメントが従業員の支払バランスに寄与している場合には、出力通貨はビジネス・グループの基本通貨でなければなりません。支払バランスは、支払方法にリンクされている支払前処理の一環として、他の通貨に変換することができます。

## コンテキストの設定

エントリが処理される前に、「給与計算」は「給与」および「エレメント・スキップ」Formula の FastFormula に必要とされるコンテキストを設定します。これには国別仕様の特定のコンテキストが含まれることがあります。すべてのコンテキストの値は、ユーザー定義コンテキスト領域（UDCA）として知られる、特別なデータ構造に保持されます。次に示す常に作成される一般的なコンテキストは、Formula に関する追加ルート情報を提供します。次のようなものです。

- ORIGINAL_ENTRY_ID
- ELEMENT_ENTRY_ID
- BUSINESS_GROUP_ID
- PAYROLL_ACTION_ID
- PAYROLL_ID, ASSIGNMENT_ID
- ASSIGNMENT_ACTION_ID
- DATE_EARNED
- ELEMENT_TYPE_ID
- TAX_UNIT
- JURISDICTION
- SOURCE_ID

国別仕様コンテキストの値を設定できるように、特別なサード・パーティ・インタフェースがコールされます。これは、US 仕様に広く用いられています。

## エレメント・スキップ・ルールの実行

エレメント・スキップ・ルールを使用することによって、エントリを処理するか否かを決定するための固有の Formula 基準を定義することができます。スキップ・ルール Formula は、Y か N の skip_flag 値を返す必要があります。

該当する場合には、スキップ Formula が起動され、（すべての通貨変換に備えて）すべての入力パラメータがメモリー内の実行結果値から取り出されます。調整のスキップを調べるときには、調整エントリ自体ではなく、通常のターゲット・エントリの入力値から Formula 入力が取り出されます。

特定のエレメントに関しては、事前定義の国別仕様の特定のスキップ・ルールがある場合もあります。この追加サード・パーティ・スキップ・フックは、内部機能が通常のスキップ Formula を検索すると同時にコールされます。この国別仕様スキップ・ルールは、'C' コードで定義されています。

### エレメント・エントリの処理モード

「給与計算」は、処理モードを使用して、エレメントのエントリを処理するかどうかを制御します。まず、処理すべきであることを示すようにモードが設定されます。次に、入力タイプと、スキップ・ルールが起動されたかどうかによっては、別のモードが設定されることがあります。これにより、現在のエントリ、そして（可能性としては）同じエレメントのその他のエントリの処理が制御されます。たとえば、上書きエントリを処理するときには、モードは「上書き」に設定されます。このモードは、このエレメントの処理中持続されるため、その他のエントリは処理されません。

## バランスの作成と保守

「給与計算」は、バランスと最新バランスにアクセスし、それらのバランスを保守する必要があります。つまり、給与計算では次の処理が行われます。

- 既存のアサイメント・レベルまたは個人レベルの最新バランスをすべてメモリーにロードします。
- ロードされたすべてのバランスの有効期限を確認し、失効している場合は 0（ゼロ）に設定します。
- 必要に応じて、メモリー内で新しく最新バランスを作成します。
- メモリー内で現在のバランスの値に適切な実行結果を追加します。
- データベースに新しいバランスを書き込みます（特定のバランス累計期間タイプのみ）。

最新バランスの詳細は、7-1 ページの「Oracle Payroll のバランス」を参照してください。

## メモリーへのバランスのロード

既存のアサイメントまたは個人レベルの最新バランス（および関連するバランス・コンテキスト）はすべて、エントリが処理される前にメモリーにロードされます。この基本データ構造は、balance_type_id によって常に順序付けされる、二重にリンクしたリストです。バランス値自体は、Oracle 番号として保持および処理されます。この場合は、2 種類のバランスが個別の表に保持されているため、フェッチは 1 つのユニオンになります。

## 最新バランスの失効チェック

最新バランスは、累計期間によって決定されるタイミングで失効します（つまり 0（ゼロ）に戻ります）。たとえば、YTD（年累計）のバランスは、年末に失効します。

ロードされているすべてのバランスで失効していないかどうかがチェックされます。失効していた場合には、0（ゼロ）に設定されます。失効ステップは、バランス・コンテキスト値を処理する必要があるため、ロード・ステップとはまったく関係ありません。

失効チェックを処理するために、「給与計算」は PL/SQL パッケージに保持されている失効チェック・コードをコールします。別の失効チェック・レベルについていくつかの前提をつけることによって、必要とされるアクセスの回数が減るため、パフォーマンスが悪くなることはありません。この前提は、バランスの失効チェック・タイプによって決定します。参照 : 失効チェック・タイプ : 7-7 ページ

## メモリー内最新バランスの作成

すべてのバランスがデータベースからロードされるわけではないので、作成する必要があるものもあります。一度作成したバランスは、保守する必要があります。

累計期間タイプによっては、新たに作成または更新されたバランスを表に書き込む必要があります。

バランスの累計期間タイプによって、給与計算による扱い方が決定します。たとえば、累計期間タイプが F のバランスは挿入されますが格納されないため、「給与計算」はメモリー内にバランスを作成します。累計期間タイプの説明は、7-6 ページの「累計期間タイプ」を参照してください。

コード中には、メモリー内バランスが作成される箇所が 3 つあります。1 つの場所は累計期間タイプ A、P および F に使用され、2 つの場所はタイプ R に使用されます。

- メモリー内バランスは、累計期間タイプが A、P または F で、まだメモリー内バランスとして保持されていない定義済バランスに Formula がアクセスしたときに作成されます。メモリー内バランスは、Formula によってアクセスされた値を使用して作成されます。
- 実行レベル・バランス累計期間タイプ（R）の定義済バランスに Formula がアクセスした場合は、Formula の実行前に 0（ゼロ）という値を持つメモリー内バランスが作成されます。（実行レベル・バランスは、定義上 0（ゼロ）である必要があります。）

- 実行レベル累計期間タイプ（R）の定義済バランスへの挿入がコードによって試みられた場合、バランス挿入時前に、0（ゼロ）という値を持つメモリー内バランスが作成されます。

前述のルールの必然の結果として、「実行レベル」累計期間タイプは別として、特定の事前定義バランスが実行された Formula によってアクセスされた場合には、最新バランスはそのバランスに対してのみ作成されます。

## メモリー内バランスへの実行結果の追加

次に、該当する実行結果が、そのバランスの現在の値に追加されます。

使用されるアルゴリズムの概要を次に示します。

1. 各処理済実行結果で、各実行結果値によって潜在的に挿入されているバランス・タイプを識別する、バランス挿入を確認します。
2. メモリー内バランスをスキャンして、なんらかの潜在的挿入がないか確認します。
3. ある場合は、バランス挿入チェックを実行します。

   バランス挿入チェックの方法は、バランス累計期間に対するバランス挿入チェック・タイプによって決定します。参照 : バランス挿入チェック・タイプ : 7-6 ページ
4. バランス挿入チェックの結果が、実行結果によるバランスの挿入を必要とするものの場合には、バランス値 = バランス値 +（実行結果 * スケール）になります。

実行レベル・バランスに挿入される可能性のある実行結果値については、バランス挿入チェックが発生する前に、「給与計算」はそれらをメモリー内に作成する必要がある場合もあります。「給与計算」はこの時点でどのバランスが必要とされるかを識別できないため、必要であると思われるバランスをすべて作成する必要があります。

実際には、これは、検査されている実行結果によって潜在的に挿入される可能性のある実行レベルの事前定義バランスごとにバランスを作成することを意味します。

---

**注意：** 累計期間タイプが R で、バランス挿入チェック・タイプが S に設定されている場合は、US 仕様に固有のケースです。この場合には、別のアルゴリズムが使用されます。

---

## メモリー内バランスの書込み

メモリー内バランスの内容（および関連するコンテキスト）は、必要に応じて、すなわち、メモリー内バランスに置換フラグが設定されている場合には、データベースに書き込む必要があります。累計期間タイプが A または P のバランスのみが書き込まれます。これは、現アサイメント処理のすべてのエントリが処理された後に発生します。

アサイメントのすべてのエレメント・エントリが処理されると、メモリー内バランス・リストがスキャンされ、データが配列バッファに移動された後で、配列がデータベース上に挿入または更新されます。

# Formula の実行

「給与計算」では、FastFormula をコールすることにより複雑な計算を実行できます。

**注意：** Formula 処理ルールにより、エレメントに対して Formula が定義されていても結果値が NULL でない場合には、その Formula は起動しません。

## FastFormula インタフェース

「給与計算」が FastFormula へのアクセスに使用するインタフェースは、次の 2 つのセクションによって構成されます。

- インタフェースの共通部分（すべての製品で使用可能）

  これは、Formula の内部データ構造から Formula に入力されるデータ（コンテキストおよび入力）や Formula からの出力（Formula 結果）に至るまでのポインタを設定します。

- 特別インタフェース

  これは、特に「給与計算」用に設計されたものであり、Formula のデータベース・アイテム・キャッシュへのアクセスを可能にします。

## 「給与計算」による FastFormula の実行

「給与計算」は次のステップから成ります。

1. 新規 Formula が実行されることを宣言します。
2. Formula は、必要とされる Formula コンテキストおよび入出力を実行コードに知らせます。
3. メモリー内バランス連鎖がスキャンされます。

   Formula は、最新バランスとして保持されている定義済バランスにアクセスした場合、そのバランスの現在の値を FastFormula データベース・アイテム・キャッシュに書き込みます。
4. Formula コンテキストがすべて満たされます。値はすべて、ユーザー定義コンテキスト領域（UDCA）から取り出されます。
5. 「入力が」変数である場合に Formula に渡される値が満たされます。これは、入力変数名と一致する、関連する入力パラメータ名を持つ実行結果値を検索することによって行われます。
6. FastFormula が実行コードに指示した出力がバッファ領域に移動されます。

## Formula の実行

サード・パーティ Post Formula フックがコールされます。これにより、国別仕様に依存する特別な関数は、「給与計算」による処理前に Formula 実行結果を処理できるようになります。たとえば、特定の実行結果を抑止することができます。

Formula 実行結果が処理されます。

## Formula 実行結果の処理

Formula の実行に続いて、「給与計算」は戻された結果を順次ループし、Formula 実行結果ルールによって要求されたとおりにそれらの結果を処理します。「給与計算」は、戻された Formula 実行結果と一致する Formula 実行結果ルール名を検索します。実行結果ルールにはいくつかのタイプがあります。次では、内部処理の観点からこれらのルールの概要について説明します。

### メッセージ・ルール

重要度レベルのメッセージが致命的である場合には、アサイメント・レベルのエラーが発生します。そうでなければ、メッセージがメッセージ表に書き込まれます。メッセージの長さが実行結果値表に保持できるサイズ（現在は、60 文字）に制限されていることに注意してください。

### 直接ルール

「表示単位」が金額の場合には、必要に応じて値が丸められます。次に、実行結果値連鎖で結果値を保持するエントリが検索され、更新されます。置換フラグがこれを示すように設定されます。

### 間接およびオーダー間接ルール

これら 2 つのタイプは、非常によく似た処理を引き起こすため、一まとめにされます。現在のエレメント・エントリの処理中はすべての間接要素が一時連鎖に保持され、後でメイン・エントリ連鎖にマージされます。

まず、一時連鎖が検索されます。そのエレメントに既存のエントリが存在していない場合には、新規エントリが作成され、連鎖に追加されます。次に、間接ルールの場合だけは、該当する入力値が検索され、新規数値に更新されます。オーダー間接ルールの場合は、間接エントリの下位優先順位が Formula 実行結果値に設定されます。

> **注意：** 2 つの Formula 実行結果ルールが同じ入力パラメータをターゲットとしている場合には、処理される 2 番目の結果が優先します。

すべての Formula 実行結果の処理に続いて、間接要素の連鎖がメイン・エレメント・エントリ連鎖の適切なポイントにマージされます。何が適切かは、メイン処理の優先順位と下位優先順位によって異なります（オーダー間接ルールを使用して設定できます）。

「給与計算」は、間接エントリの処理優先順位がその間接エレメントを引き起こしたエレメントと同じにならないようにします。しかし、フォームではこれは許されません。同じ優先順位を持つ間接要素が、特に US 国別仕様の要求に応じて提供されています。

しかし、同じ優先順位を持つ間接要素は、エンドレス・ループを生成するため、問題の原因となる可能性があります。

#### 固定処理更新ルール

「給与計算」は、更新の対象となるエレメント・エントリを見つけ出すために、PL/SQL プロシージャをコールします。このプロシージャは、有効日更新を実行します。このエントリが偶然にもそのエントリ連鎖のはるかに下の方に存在していた場合には、その値が変更を反映するように更新されます。

#### 固定処理停止ルール

「給与計算」は、停止の対象となるエレメント・エントリを見つけ出すために、PL/SQL プロシージャをコールします。このプロシージャは、有効日削除を実行します。

## 実行結果の処理

実行結果とそれに関連する実行結果値は、エレメント・エントリおよびエレメント・エントリ値の必然の結果です。エントリがいくつかのエレメントに対する適格性を表すのに対して、結果と値にはそれらのエントリを処理した後の効果が含まれます。

処理中は、実行結果と値はメモリー内に保持され、メモリー内エレメント・エントリ連鎖から離されます。これは、データベース項におけるそれらの密接な接続を反映しています。

## 実行結果と実行結果値の作成

実行結果と値は、次の 3 つの方法のいずれかを使用して内部生成されます。

- エントリおよび入力値がロードされたときにロードされる - 変動処理エレメント・エントリから発生した挿入前実行結果として。
- なんらかの欠落実行結果と値がある場合に、該当するエレメント・エントリを処理する前に、「給与計算」により作成される。
- 間接結果により作成される。

## 実行結果値のデフォルト設定

「給与計算」は、ホット / コールド・デフォルト設定を処理する一方で、実行結果と値が存在しているかチェックします。実行結果と値がすでに存在している上に、それらが NULL である場合には、「給与計算」はそれらをデフォルトに設定しようとします。

通貨換算が必要な場合は、同時に実行されます。内部的には、その計算には Oracle 番号が使用されます。この後に、「給与計算」は、表示単位が「金額」の入力パラメータを処理している場合には、必要に応じて実行結果の端数処理を実行します。

## 実行結果と値のデータベースへの書込み（フラッシュ）

このプロセスでは、実行結果と値が特別のバッファに移動された後で、実行結果と値がデータベースに書き込まれます（更新または挿入）。これには、配列処理技法（最新バランスで使用されるのと同じ技法）が用いられます。

このプロセスは通常、実行結果のフラッシュと呼ばれます。これがトリガーされる状況は 2 つ考えることができます。

- メモリー内に保持されていないデータベース・アイテムにアクセスする Formula を実行しようとしている場合。このデータベース・アイテムのルートは、「給与計算」自体で今までに生成された実行結果にアクセスする必要のある場合があります。このような仮定は、確実に見つけ出す方法がないためになされています。
- そのアサイメント処理のすべてのエレメント・エントリが処理されると、残りの実行結果と値がすべてフラッシュされます。

## 給与データ・キャッシュ

処理中は、「給与計算」は静的定義データを表すいくつかのエンティティの属性にアクセスする必要があります。たとえば、特定の入力パラメータのエレメント名またはバランス挿入を認識する必要がある場合もあります。さらに、同じデータは、通常、繰り返しのアクセスを必要とします。このようなデータが必要とされる度にデータベースから選択された場合には、深刻なパフォーマンスの劣化が生じる恐れがあります。

このような問題を解決するために、特別な静的給与データ・キャッシュが搭載されています。最初にアクセスされたときに、エンティティの該当するすべてのデータがメモリーにロードされます。それ以降は、そのデータへの後続のアクセスはすべて、メモリーに直接行きます。

# 支払前処理

「支払前処理」は、支払のために「給与計算」によって生成された支払を準備します。また、各アサイメントの支払を準備し、アサイメントの各支払方法の PAY_PRE_PAYMENTS にその結果を挿入します。

「支払前処理」では、次のことも行われます。

- アサイメントの各支払方法によって支払う金額を計算し、支払方法が外国通貨の場合には、通貨変換します。
- 第三者支払の準備を行います。（JP 以外）

  たとえば、債権差押え通告、裁判所命令および養育費。第三者支払は、従業員に対して特別支払方法を定義することによって管理されます。

## 支払方法の設定

導入中に、振出口座詳細を使用して独自の特別支払方法を設定します。従業員を雇用するときに、その従業員に対して 1 つ以上の支払方法を記録し、% または金額で支払を割り当てることができます。支払方法を各種通貨で記録することも可能です。

「支払前処理」は、各アサイメントの支払方法に従って支払を準備します。Oracle Payroll で処理できるのは、次の 3 種類の事前定義支払タイプです。

- 小切手（JP 以外）
- 磁気テープ（NACHA/BACS など）
- 現金（UK、JP のみ）

ご自分のビジネス・ニーズを満たすのに必要な数だけの支払方法を、（3 種類の事前定義の支払タイプを基に）設定することができます。

すべての給与システムでデフォルトの支払方法が指定されています。特定のアサイメントに関して個人の支払方法が指定されていない場合には、支払前処理では、デフォルトの方法が使用されます。

---

**注意：** 磁気テープ・タイプをデフォルトの支払方法とすることはできません。これは、磁気テープ支払方法では、US の場合の事前通知詳細など、従業員の銀行口座詳細に関する知識が必要とされるためです。5-18 ページの「事前通知（US のみ）」を参照してください。

---

支払方法は、アサイメントの支払方法の優先順位に基づいて処理されます。たとえば、従業員が次のような支払を希望する場合があります。

1. 磁気テープ支払による給与の 50%の銀行口座振込み
2. 小切手による 100 ドルの支払
3. 現金による 100 ドルの支払

「支払前処理」は、支払金額ですべての支払を賄える場合には、優先順位に基づいて支払を準備します。支払金額が支払方法によって指定される額を下回る場合には、システムは 100%まで支払って、停止します。支払金額が支払方法によって指定される額を上回る場合には、システムは超過分を最新支払方法に追加します。

## 現金支払の準備（UK、JP のみ）

Oracle Payroll を使用して現金支払の準備をしている場合は、各従業員の銀行券および貨幣要件を計算することができます。「支払前処理」は、金額を個別の支払通貨単位に分け、その結果を PAY_COIN_ANAL_ELEMENTS 表に挿入します。

Oracle Payroll によって管理される現金支払に関しては、支払う各通貨の通貨単位を定義することができます。また、通貨の各金種の最低数を指定するための通貨間分析ルールを定義することも可能です。

### キャッシュ・ルールの設定

キャッシュ・ルールの設定は、次の 2 つのステップから成ります。

1. パッケージ本体 hr_cash_rules を変更する

   この変更により、設定するキャッシュ・ルールの名称をテストしてから、支払が実行されるようになります。たとえば、ルール名が 'TENS AND FIVES' である場合には、次のように入力します。

```
if cash_rule = 'TENS AND FIVES' then

--

hr_pre_pay.pay_coin(6, 10)

hr_pre_pay.pay_coin(3, 5)

--

-- number to pay ---^ ^--- unit value of currency

--

--end if;\
```

   通貨ドルに関してこのキャッシュ・ルールを適用した場合には、最低 6 枚の 10 ドル紙幣と 3 枚の 5 ドル紙幣が支払われることになります（十分な財源があった場合）。

2. ルールを登録する

- 「参照値」ウィンドウに移動し、CASH ANALYSIS の参照タイプを問い合せます。
- 内容および摘要フィールドを TENS AND FIVES に設定して、新規キャッシュ・ルールを追加します。
- 組織の支払方法を設定するときにこのキャッシュ・ルールを使用します。

## 事前通知（US のみ）

事前通知検証（照会としても知られる）は、磁気テープ・タイプの支払方法に適用されます。この検証は、支払を行う前に銀行詳細をチェックする必要がある場合に実行されます。たとえば、従業員が銀行または銀行詳細を変更した場合には、従業員の銀行口座に対して 0（ゼロ）の値の支払が行われます。その後の支払は、後続の方法またはデフォルトの方法によって行われます。

## 連結セット

「支払前処理」は、連結セットに対して実行されます。連結セットは複数の処理グループを 1 つに結び付けるタグです。連結セットを使用して、まだ支払前処理が行われていないセットの中のすべてのアサイメント処理の支払前処理を行うことができます。異なる給与システムや異なる期間のアサイメント処理であってもかまいません。たとえば、連結セットを使用して、会社の月次および週次の両方の給与を磁気テープ処理により支払うことができます。

## 第三者支払（JP 以外）

第三者支払は従業員の給与の課税後の控除額であり、組織または個人に対して支払われます。たとえば、裁判所命令は地方裁判所に支払うべきものですが、養育費支払命令は配偶者またはその他の個人に対して直接支払うべきものです。

これらの支払は、わずかに異なる方法で処理されます。その支払に対して実行結果値をもたらすエレメント・エントリは、使用する支払方法の詳細を保持しています。これにより、1 つのアサイメントで第三者支払エレメントのエントリを複数作成し、それぞれのエントリで異なる支払先を表すことができます。たとえば、従業員は、養育費という第三者エレメントを 2 人の異なる人物に支払うことができます。

第三者支払は、磁気テープまたは小切手でのみ行うことができます。現金支払はできません。さらに、これらの方法によって支払の全額が支払われるため、1 つの方法のみが用いられます。これらの支払に関してはデフォルト方法がないので、必ず支払方法を指定する必要があります。US: 磁気テープ照会検証が失敗した場合には、そのアサイメントに対してエラーが生成されます。

## 換算レート

「支払前処理」は、支払が報酬バランスの通貨（第三者支払の場合は、エレメント出力通貨）と異なる通貨で行われる場合には、通貨変換計算を行います。この処理で 2 種類の通貨の換算レートを検出できなかった場合には、そのアサイメントに対してエラーが生成されます。

## 支払方法の上書き

支払前処理を実行するときに、「支払方法の上書き」を指定することができます。この方法では個人支払方法が上書きされるため、全額が上書きされた方法で支払われます。唯一の例外は第三者支払です。第三者支払は、エレメント・エントリで指定された方法に従って支払われます。

「支払方法の上書き」は、次のいずれかを取ることができます。

- 現金
- 小切手（JP 以外）

上書き方法として磁気テープ支払を指定することはできません。これは、このタイプの支払には銀行口座詳細に関する前知識が必要とされるためです。

## プロセス

「支払前処理」は、給与計算処理とアサイメント処理を作成します。アサイメント処理は、支払前処理と連動しない指定された給与計算 / 連結セットのアサイメント処理に基づいています。連動によって、「支払前処理」がロールバックされるまでは「給与計算」がロールバックされないようにします。このため、新規アサイメント処理は、給与計算のアサイメント処理と連動して作成されます。

参照 : アサイメント・レベル連動 : 5-61 ページ

### チャンク

アサイメント処理は、PAY_ACTION_PARAMETERS 表の CHUNK_SIZE 処理パラメータによって示されるサイズである、チャンクと呼ばれるグループに分割されます。このプロセスでは、THREAD 処理パラメータに応じて、いくつかのスレッド（子プロセス）が起動されます。各スレッドは、処理するチャンクを取り出し、アサイメント処理を実行した後で、すべてのチャンクが処理されるまで別のチャンクを取り出し続けます。多数のスレッドを使用することによって、マルチプロセッサ・マシンのパフォーマンスの向上を図ることができます。

## PL/SQL プロシージャ

C プロセスのメイン部分（支払を実行する部分）は、PL/SQL プロシージャのハーネスです。PL/SQL プロシージャは支払前処理表のエントリを作成します。

スレッドは、次のようにアサイメントを処理します。

- 第三者詳細を取り出し、個人支払方法の定義に従って、第三者支払を記録する。
- PL/SQL バランス関数を使用して、アサイメントの報酬バランスの値を取り出す。
- 支払方法の定義に従って、この値の支払を記録する。

## エラー処理

処理中には、次の 2 種類のレベルでエラーが検出される可能性があります。

- 給与計算処理レベル

  これらのエラーは致命的です。

- アサイメント・レベル

  これらのエラーは、アサイメント処理の実行中に発生します。このレベルでエラーが検出された場合には、アサイメント処理のステータスに「エラー」とマークが付けられ、処理が続行されます。処理が完了した場合には、給与計算処理ステータスに「完了」とマークが付けられます。

MAX_ERRORS_ALLOWED 処理パラメータを使用することにより、いくつのアサイメント・エラーが処理されると、給与計算処理レベルでエラーが発行されるかを設定できます。MAX_ERRORS_ALLOWED が見つからない場合には、チャンク・サイズがデフォルトとして使用されます。

エラー・メッセージはすべて、PAY_MESSAGE_LINES 表に書き込まれ、詳細説明がログ・ファイルに記録されます。

このようなエラーの処理方法を使用することによって、検出されたエラーが比較的重要でない場合には、「支払前処理」は処理を続行します。たとえば、「支払前処理」で何千ものアサイメントを処理する場合、そのうちのいくつかのアサイメントが現金で支払われても、通貨詳細がロードされていない場合には、現金支払の伴うアサイメントに対してエラーが生成されます（「キャッシュ・ブレークダウンは実行できません」）。ほとんどのアサイメント処理は完了しますが、エラーのあるアサイメントのみ再実行する必要があります。

# 支払処理

支払の結果を準備するには（支払方法に従って）支払前処理を実行した後、従業員の支払を作成します。

Oracle Payroll では、次の 3 種類の支払処理を実行することができます。

- 磁気テープ処理 - MAGTAPE

  参照 : 磁気テープ処理 : 5-21 ページ

- 小切手処理 - CHEQUE（JP 以外）

  参照 : 小切手ライター処理（JP 以外）: 5-41 ページ

- 現金支払処理 - CASH（UK、JP のみ）

  参照 : 現金処理 : 5-49 ページ

これらの支払処理では、各支払タイプに割り当てられた未払い支払前処理値が呼び出され、必要な支払ファイルが作成されます。

特定の従業員に対して行う手動支払もすべて記録することができます。これらの支払は支払処理では扱われません。手動支払を記録することにより、支払前処理は支払済となります。

# 磁気テープ処理

「磁気テープ処理」は、支払期日を生成し、その日付を磁気テープ上のファイルに書き込みます。支払銀行に渡されるのはこのテープです。

磁気テープ・ファイルには次の 2 種類があり、それぞれに作成方法が異なります。

- 支払
- 年末税レポート

これらのテープの実際のフォーマットは、国別仕様になります。

テープ処理は、Oracle ストアド・プロシージャや FastFormula Formula をコールし、必要なテープ・ファイルを作成する単純な 'C' ハーネスです。このルーチンは汎用で、磁気テープ・レポートを必要とするすべてのタスクに使用できます。テープの実際の構造や内容は、格納されているプロシージャや一連の Formula によって完全に定義されています。

このルーチンを使用するいくつかの例を次に示します。

- BACS
- NACHA
- W2
- P35 発行（およびその他の国におけるこれと同等のもの）

**注意：** 磁気ファイル内のエントリの順序が重要となります。このため、「磁気テープ処理」をマルチ・スレッドで実行することはできません（「支払前処理」や「小切手ライター処理」とは異なる）。

**関連項目：** 給与アーカイブ・レポータ（PAR）処理 : 6-1 ページ

# 磁気テープ支払処理の実行

給与計算アサイメント処理作成コードは、「磁気テープ支払処理」へのエントリ・ポイントです。従業員の磁気テープ支払は、関連する支払前アサイメント処理と連動する、給与計算処理およびアサイメント処理としてオラクル人事管理システムに記録されます。連動によって、磁気テープ処理が存在している間は支払前処理がロールバックされないようにします。

第三者支払（会社の健康保険制度の分担金など）によって給与計算処理やアサイメント処理が実行されることはないため、第三者支払では磁気テープ・レポート・インタフェースが使用されます。

## バッチ処理パラメータ

次のパラメータを使用して PYUGEN を実行します。

**consolidation_set_id** **必須**

どのセットの未払いの支払前処理を支払うかを定義します。

**payment_type_id** **必須**

PL/SQL 駆動プロシージャを定義します。

**effective_date** **オプション**

処理の有効日を識別します。

**payroll_id** **オプション**

処理対象のアサイメントを、有効日の、指定された給与計算のものに限定します。

**start_date** **オプション**

どのくらい前まで遡って支払前処理を検索するかを指定します。このパラメータが指定されないと、時間の初めに遡ってスキャンされます。

**organisation_payment_method_id** **オプション**

その支払の未払いの支払前処理と連動するアサイメント処理を作成します。

**legislative** **オプション**

自由書式で、すべての給与計算処理に使用できるパラメータです。ローカライズ・チームはこれらのパラメータを使用して、エンティティ水平線によって給与計算処理にアクセスできるようにされている、多数の国別仕様パラメータを渡すことができます。

## 支払タイプ用 PL/SQL プロシージャ

システムは、データベース上で支払タイプに対して指定された PL/SQL 駆動プロシージャを使用します（たとえば、<package name>.<procedure name>）。「磁気テープ・ライター処理」用 PL/SQL プロシージャは、アサイメント処理を駆動しないでください。処理されたアサイメントをさらに限定しないようにしてください。アサイメントをさらに限定すると、いくつかの磁気テープ・アサイメント処理が未処理のまま残される危険性があります。プロセスが初めて PL/SQL を実行した場合に渡されるパラメータの 1 つが、給与計算処理 ID（PAYROLL_ACTION_ID）です。

「磁気テープ処理」は、磁気テープ処理の有効日またはそれ以前の支払前処理を実行します。磁気テープの有効日のデフォルトは、AOL 環境においてはセッション日となり、AOL 以外の環境においては sysdate になります。

## 出力ファイル名

生成される磁気テープ・ファイルには、次のような通常のファイル命名規格に従って名前が付けられます。

p<trunc(conc_request_id, 5)>.mf

要求 ID が 5 文字より短い場合には、ファイル名は 0（ゼロ）で埋められます（例 :p03451.mf）。

$ APPLCSF が定義されている場合は $APPLCSF/$APPLOUT ディレクトリに書き込まれ、それ以外の場合には $PAY_TOP/$APPLOUT に書き込まれます。

この処理によって、その他のファイルもいくつか作成されることがあります。これらのファイルを使用して、処理中のアサイメントを監査できます。監査ファイルは、ファイル拡張子が .a<file_number> となる以外は、同じ方法で作成されます。このため、Formula が監査ファイル 6 の値を返した場合、拡張子 .a6 を持つファイルが、前述のようなコンカレント要求 ID を使用して適切なディレクトリに作成されます。

# 磁気テープ・レポートの実行

磁気テープ・レポートは、給与計算処理やアサイメント処理としては記録されません。エントリ・ポイントは、固有の磁気テープ・コード、PYUMAG です。処理されるアサイメントは PL/SQL により決定します。

**必須パラメータ：**

- PL/SQL 駆動プロシージャ（<package name>.<procedure name>）
- 出力ファイル（フルパス名を含む）

**オプション・パラメータ：**

- 監査ファイル接頭辞（拡張子の接頭辞＋フルパス）
- 有効日（PL/SQL 駆動プロシージャに対するパラメータ）

PL/SQL に対するオプション・パラメータは、汎用テープ・ライター処理がパラメータ名やパラメータ値の PL/SQL 表に挿入できるように、トークン化する必要があります。これらの表は、汎用ライター処理と PL/SQL 駆動プロシージャの間のインタフェースを構成します。

参照 : PL/SQL 駆動プロシージャ : 5-28 ページ

磁気テープ処理は、磁気テープ処理の有効日またはそれ以前の有効日を持つ Formula のみを処理します。磁気テープの有効日は、AOL 環境においてはセッション日を、AOL 以外の環境においては sysdate を、それぞれデフォルトとして取ります。

## 出力ファイル名

磁気テープ・ファイル名は、指定されない場合には生成されます。このファイルは次のようなフォーマットになっています。

o<trunc(conc_request_id, 5)>.mf

監査ファイルの接頭辞が設定されていないときにプロセスが監査ファイルに書き込もうとした場合には、コンカレント要求 ID が接頭辞として、また .out が拡張子としてそれぞれ使用されます。こうした状況のもとでは、すべての監査戻り値がこのファイルに書き込まれます。

## SRS 定義

汎用テープ・ライター処理は SRS を使用して、実行可能ファイルとして一度定義されます。その後は、実行可能ファイルを呼び出すコンカレント・プログラムを任意の数のみ定義できます。各コンカレント・プログラムは、独自のパラメータ・セット、独自の隠しパラメータ、デフォルトなどを持つことができます。たとえば、次の 2 種類のコンカレント・プログラムを定義できます。

- W2 レポート
- イリノイ州四半期州税レポート

両方とも、磁気テープ・ライター実行可能 PYUMAG を使用します。また、適切な PL/SQL プロシージャを指定する隠しパラメータはもちろん、固有のパラメータもそれぞれ持っている可能性があります。これらは、ユーザーにはまったく別個のレポートのように見えます。これは、SRS プロセス・インタフェースで設定されます。

同様に、磁気支払もユーザーにまったく別個のプロセスのように見えるようにすることが可能です。支払タイプが隠しパラメータであるため、汎用コードはそれから PL/SQL 駆動プロシージャを決定するというのが、唯一違う点です。

## 磁気テープ処理の仕組み

磁気テープは、通常、次の 2 つに分けられます。

- レコード
- フィールド

テープにレコードが書き込まれる順序は、厳密に定義されたルールに準拠しています。このため、次に書き込むレコードの名前をテープに戻すコードの一部分を記述することも可能です。

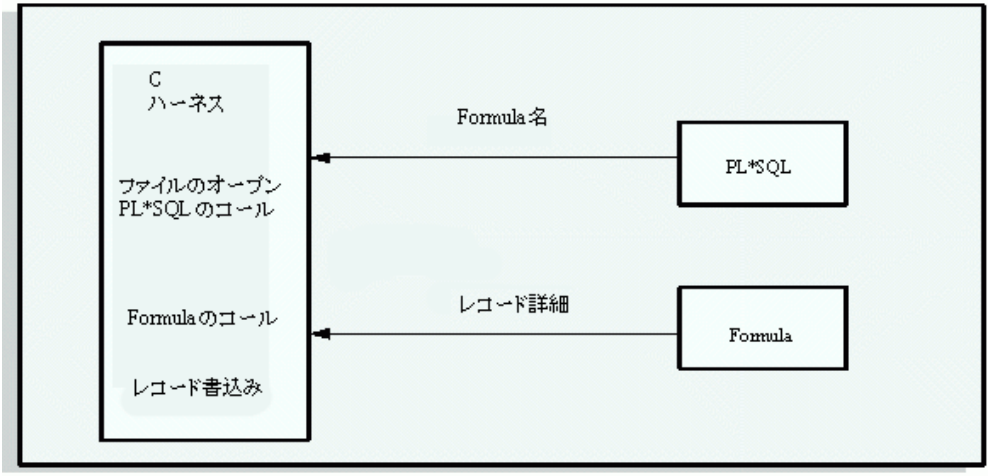
同様に、実際のレコードのフィールド領域や長さの要件は厳しくなっています。例 :

| レコード | フィールド |
|---|---|
| テープ・ヘッダー | バッチ ID、会社名、バッチ・レコード長など |
| 従業員 | 従業員 ID、給与、年齢、役職など |
| テープ・フッター | 処理済レコード数、給与合計など |

## C ハーネス、PL/SQL と Formula

次の図は、「磁気テープ処理」を示したものです。

**図 5-1 磁気テープ処理**




C コード・ハーネスは、ファイル処理（ファイルのオープン / クローズ、ファイルへの書込み）を実行し、PL/SQL や Formula がインタフェースできるようにします。

PL/SQL 駆動コードは、Formula の名称を戻して、レコードを順番にそろえます。

各 Formula は、テープにテープ・ヘッダーを書き込むなど、1 種類のレコードを書き込みます。これによりレコードの内容が定義されます。

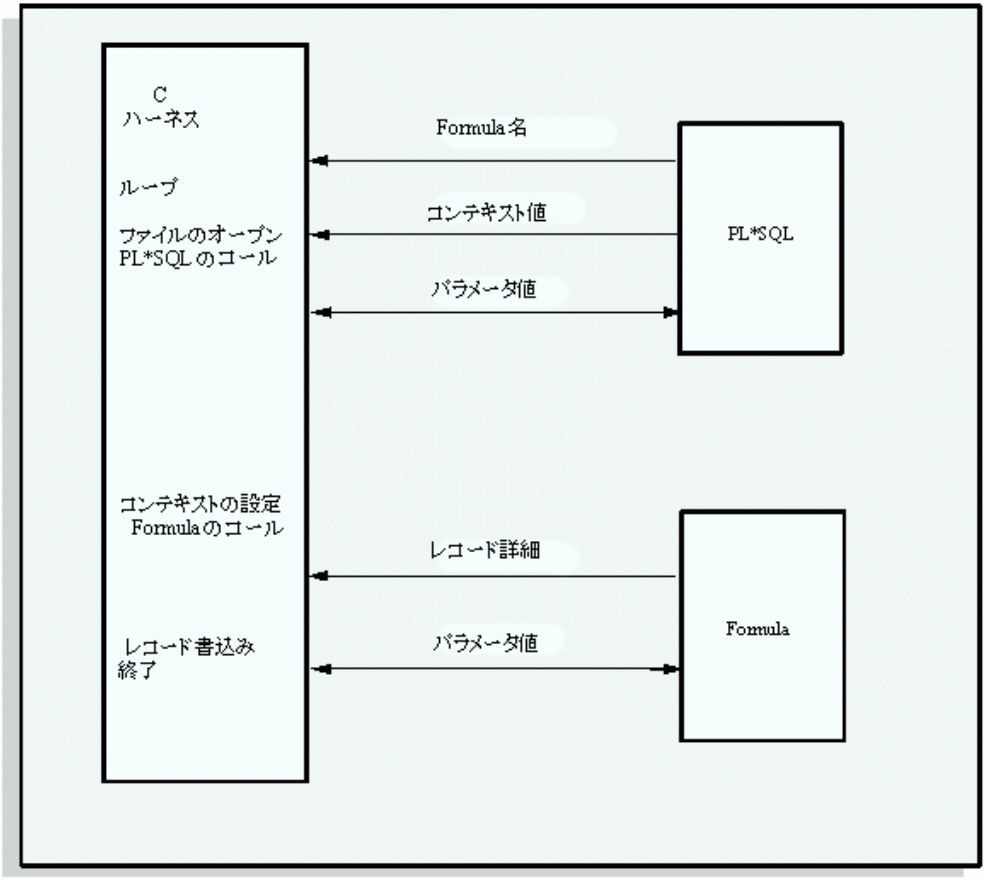
この Formula とレコード名を取得し、レコードをテープに書き込むという処理は、すべてのレコードが処理されるまで繰り返されます。

## コンテキストとパラメータ値

駆動される PL/SQL は、処理のいずれの段階においても、どのタイプのレコードが必要かを決定し、コンテキスト値とパラメータ値を使用して Formula と通信します。

次の図は、C コードが PL/SQL と Formula の間のインタフェースとしてどのような働きをするか、さらにはデータがどのようにしてコンテキスト値として渡されるかを示したものです。

**図 5-2 C コード・インタフェース**




### コンテキスト値

Formula は、データベース・アイテムを使用して変数値を参照します。たとえば、従業員やアサイメント番号は、Formula やレコードを実行する度に違う値を取ることができます。

データベース・アイテムは、データベース内に保持されており、SQL 文を構成するコンポーネントから成ります。Formula の実行ごとに違う値を取ることができるので、文の WHERE 句がわずかに異なります。これは、主要値を、必須値を一意に選択する WHERE 句に置き換えることによって行われます。これらの置換値は、コンテキスト値として知られています。

コンテキスト値は、値を PL/SQL 表に挿入する「PL/SQL 駆動プロシージャ」によって設定されます。PL/SQL 表は、C コードに渡し返され、C コードによって Formula 構造に入れられます。

### パラメータ値

パラメータ値は、Formula と PL/SQL の間で転送される変数データを格納するために使用されます。たとえば、実行合計は、このようにして Formula に渡されます。

これらのパラメータでは、次のことが可能です。

- コマンド行から C プロセスに渡すことができる。
- 「PL/SQL 駆動プロシージャ」によって作成することができる。
- Formula によって作成することができる。

「PL/SQL 駆動プロシージャ」と Formula のみ、値を更新できます。

## PL/SQL 駆動プロシージャ

PL/SQL 駆動プロシージャは、磁気テープ・ファイルのフォーマットを決定します。このプロシージャは、カーソルを取り出すたびに特定の Formula を処理するカーソルをオープンすることによって、0 から作成できます。汎用 PL/SQL を使用することも可能です。汎用 PL/SQL は、磁気テープ・バッチ表を駆動しません。

'C' プロセスと格納されているプロシージャの間のインタフェースは、PL/SQL 表を広範囲に活用します。PL/SQL 表は、整数索引値によってアクセスされる単一列の表です。これらの表内の項目は、1 で始まりエレメントの数まで連続的に増加する索引を使用します。索引番号は、名前や値の表の項目を照会するのに用いられます。

PL/SQL プロシージャでインタフェースをとるのに使用される表の名前は次のとおりです。

- pay_mag_tape.internal_prm_names
- pay_mag_tape.internal_prm_values
- pay_mag_tape.internal_cxt_names
- pay_mag_tape.internal_cxt_values

最初の 2 つの表（pay_mag_tape.internal_prm_names と pay_mag_tape.internal_prm_values）は、PL/SQL と Formula にパラメータ詳細を渡すのに用いられます。これらの表は、パラメータ表内の全エントリと実行される Formula ID 用に確保されています。残りの 2 つの表（pay_mag_tape.internal_cxt_names と pay_mag_tape.internal_cxt_values）は、Formula にデータベース・アイテムのコンテキスト・ルールを設定するのに用いられます。これらの表は、コンテキスト表内の全エントリ用に確保されています。

# 汎用 PL/SQL

「磁気テープ処理」は、カーソル名が含まれているいくつかの表を駆動しない汎用 PL/SQL を使用します。これらのカーソルおよび表は、磁気テープのフォーマットを制御します。

これらのカーソルは、次の 3 種類のタイプのデータを取り出します。

- 後続のカーソルに使用されるデータ
- コンテキスト値データとして使用される必要があるデータ
- パラメータ / 変数データとして保持されるデータ

## 例

例として、2 種類の SELECT 文を次に示します。

```
cursor business is
select business_group_id,
'DATE_EFFECTIVE=C', effective_start_date
from per_business_groups

cursor assignment is
select 'ASSIGN_NO=P', assignment_id
from pay_assignments
```

前述の例では、最初の選択（DATE_EFFECTIVE）は後続の Formula に渡されるコンテキスト値です。後続のカーソルで使用するために、business_group_id 列が取り出されます。この列には、後述の関数を使用してアクセスされます。

2 番目の選択（ASSIGN_NO=P）はパラメータとして使用されます。

カーソルがオープンされると、カーソルは検索表に行を割り当て、そこに選択して挿入します（行数は、カーソルによって取り出される列数によって決まります）。たとえば、上述のカーソルが使用され、前例が実行された場合には、検索表は次のようになります。

| 最初の実行後 | 2 回目の実行後 |
|---|---|
| 50000 | 50000 |
| DATE_EFFECTIVE= | DATE_EFFECTIVE=C |
| 16-MAR-1997 | 16-MAR-1997 |
| | ASSIGN_NO=P |
| | 50367 |

## データにアクセスする関数

カーソルの中には、以前に選択されたデータにアクセスする必要のあるものがあります。これには 2 つの方法を使用できます。

- 列がコンテキストまたは個別の列として選択されている場合には（前の例のビジネス・グループなど）、get_cursor_return 関数を使用します。SELECT 文でカーソル名と列位置を指定することにより、値が戻されます。たとえば、前述の SELECT 文のビジネス・グループを取り出すには、次のコマンドを使用します。

  ```
  pay_magtape_generic.get_cursor_return('business', 1)
  ```

- あるいは、パラメータとして値を選択し、パラメータ名の指定によりこの値を検索する関数にアクセスします。たとえば、ASSIGN_NO パラメータ値を取り出すには、次のコマンドを使用します。

  ```
  pay_magtape_generic.get_parameter_value('ASSIGN_NO')
  ```

## コンテキストとパラメータ・データ

Formula は次の 2 種類のデータを必要とします。

- コンテキスト
- パラメータ

コンテキスト・データは、PL/SQL 表に保持されています。前述のように、PL/SQL 表には、PL/SQL によって、カーソルによって取り出されたデータが入れられます。コンテキスト・ルールは、下位レベルのカーソルがそのコンテキスト名に関して別の値を取り出さない限り、下位レベルに継承されます。PL/SQL は、特定のコンテキストに対しては、常に下位レベルのコンテキスト値を使用します。たとえば、前述の 2 番目のカーソルが DATE_EFFECTIVE のコンテキスト値を取り出した場合、カーソルがクローズするまで、この値が Formula で使用されます。これは、検索表の中で前の DATE_EFFECTIVE の下位レベルになります。カーソルがクローズすると、検索表内の行が再生され、DATE_EFFECTIVE コンテキストは元に戻ります。

パラメータ・データも表に保持されていますが、コンテキスト値とは異なり、値はレベルに依存していません。Formula は、入力行上のパラメータを選択して、これらの値にアクセスできます。Formula がこのパラメータの値を戻す場合は、表中のエントリが上書きされます。Formula が存在しないパラメータを戻した場合には、戻されたパラメータが表に入力されます。

## カーソル / ブロック表

パッケージ・プロシージャの駆動構造は、次の 2 種類のデータベース表に保持されています。

- PAY_MAGNETIC_BLOCKS
- PAY_MAGNETIC_RECORDS（Formula/ レコード表、下記参照）

PAY_MAGNETIC_BLOCKS 表は次のようになります。

| 名称 | NULL? | タイプ |
|---|---|---|
| MAGNETIC_BLOCK_ID | NOT NULL | NUMBER (9) |
| BLOCK_NAME | NOT NULL | VARCHAR2 (80) |
| MAIN_BLOCK_FLAG | NOT NULL | VARCHAR2 (30) |
| REPORT_FORMAT | NOT NULL | VARCHAR2 (30) |
| CURSOR_NAME | | VARCHAR2 (80) |
| NO_COLUMN_RETURNED | | NUMBER (5) |

### 例

| block_id | cursor_name | block_name | no_of_select_値 | main_block | タイプ |
|---|---|---|---|---|---|
| 1 | company_curs | companies | 2 | Y | CA |
| 2 | employee_curs | employees | 2 | N | CA |
| 3 | assignment_curs | assignments | 1 | N | CA |

- block_id はシステム生成です。
- no_of_select_values は、cursor_name によって指定された SELECT 文によって取り出された列の数です。
- main_block は、使用する開始ブロックを指定します。1 つのレポートでは、これらのいずれか 1 つしか Y に設定できません。
- タイプは、SELECT 文が表すレポートのタイプを示します。

## Formula/ レコード表

PAY_MAGNETIC_RECORDS 表は次のようになります。

| 名称 | NULL? | タイプ |
|---|---|---|
| FORMULA_ID | NOT NULL | NUMBER (9) |
| MAGNETIC_BLOCK_ID | NOT NULL | NUMBER (9) |
| NEXT_BLOCK_ID | | NUMBER (9) |
| LAST_RUN_EXECUTED_MODE | NOT NULL | VARCHAR2 (30) |
| OVERFLOW_MODE | NOT NULL | VARCHAR2 (30) |
| SEQUENCE | NOT NULL | NUMBER (5) |
| FREQUENCY | | NUMBER (5) |

### 例

| formula_name | block_ id | seq | next_ block | frequency | O/F | exec.last |
|---|---|---|---|---|---|---|
| formula 1 | 1 | 1 | - | - | N | N |
| formula 2 | 1 | 2 | 2 | - | N | N |
| formula 3 | 2 | 1 | - | - | N | N |
| formula 4 | 2 | 2 | 3 | - | N | N |
| formula 5 | 3 | 1 | - | - | N | N |
| formula 6 | 2 | 3 | - | - | N | N |
| formula 7 | 1 | 3 | - | - | N | N |

Formula/ レコードのタイプとしては、次の 3 種類の一般的なタイプが可能です。

- カーソルによって戻されたすべての行に対して実行される標準 Formula
- x 行ごとに 1 回実行される中間 Formula
- 前の Formula の結果に応じて実行される Formula（オーバーフロー Formula）

表の列は次のようになります。

- block_id は、この Formula がその一部となっているブロックを意味します。
- seq は、ブロック中の順序を意味します。
- next_block 列は、この Formula が実行された後に、next_block よって定義されたカーソルをオープンし、そのカーソルに関する行がなくなるまでそのブロックの Formula を実行することを示します。
- frequency 列は、中間 Formula が、Formula の実行前にスキップする行数を指定するのに使用します。

- O/F（オーバーフロー）は、Formula がオーバーフローかどうかを指定します。オーバーフローである（Y に設定される）場合、そして最後の Formula が TRANSFER_RUN_OVERFLOW フラグを Y に設定して戻した場合には、Formula が実行されます。

  同様に、Formula が繰返しオーバーフロー（R に設定）、TRANSFER_RUN_OVERFLOW フラグが Y に設定されている場合には、Formula が Y に設定された TRANSFER_RUN_OVERFLOW を返さなくなるまで、その Formula は繰り返され続けます。

- exec.last 列は、あらゆるタイプの Formula に適用されますが、最も一般的なのが中間 Formula です。この列は、最後の行がカーソルによって取り出された後に、もう 1 度その Formula を実行することを指定します。

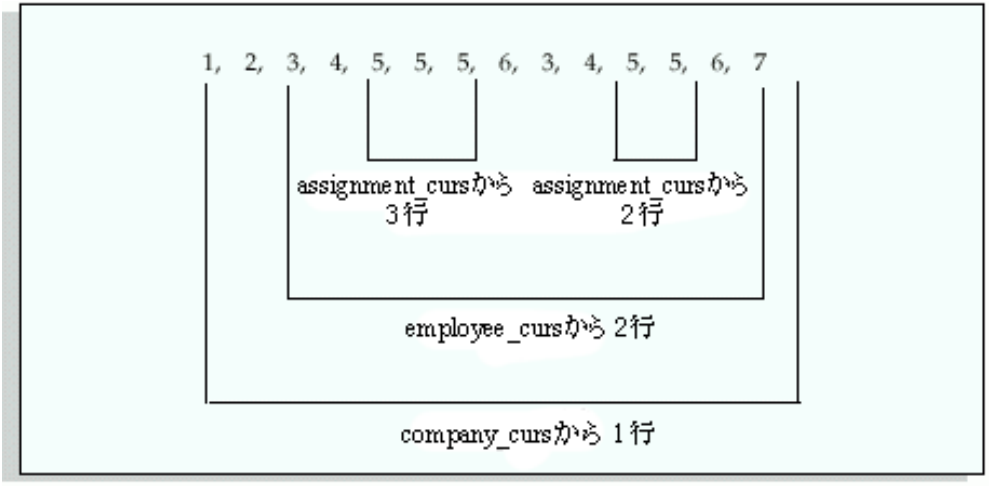
  中間 Formula に関しては、この列を次の 4 種類の値に設定することができます。

  – N - 最後の行が戻された後は、絶対に実行しない。
  – A - 最後の行が戻された後も、常に実行する。
  – R - このカーソルに対して中間 Formula が実行された場合にのみ実行する。
  – F - これがこのカーソルに対する Formula の最初の実行である場合にのみ実行する。

  **注意：** オーバーフローおよび標準 Formula に関しては、N と A のみが有効です。

上述の仕様を用いれば、次の順序で Formula を取り出すことができます。

**図 5-3 Formula の順序付け**




汎用 PL/SQL プロシージャは、処理するレポートのタイプを指定します。これは、プロセスのコール時に、パラメータ MAGTAPE_REPORT_ID を渡すことによって行われます。上図は、プロセスのコール時に、MAGTAPE_REPORT_ID=CA がどのように渡されるかを示したものです。

## Formula インタフェース

通常、磁気テープは多数のレコード・タイプから構成されます。Formula を各レコード・タイプ（生成）と関連付けることをお薦めします。Formula は次のことを行います。

- レコードにおけるフィールド位置の定義
- 計算の実行
- テープに書き込まれた詳細のレポート作成（監査）
- 各種レベルのエラー・メッセージの呼出し

PL/SQL ストアド・プロシージャは、メイン制御フローを提供すると同時に、Formula がコールされる順序を決定します。

このルーチンでは、FastFormula を使用してレコードが準備されます。レコードは、磁気テープへの転送に備えて、ASCII ファイルに書き込まれます。必要な処理をインプリメントするために、さらに多くの Formula 結果ルール・タイプが用意されています。それらのタイプを次に列挙します。

| | |
|---|---|
| **転送** | これは、出力パラメータを格納されているプロシージャの入力に転送します。パラメータは、Formulaの次の実行に使用される前に、格納されているプロシージャによって変更される場合もあれば、変更されない場合もあります。 |
| **WRITE TO TAPE** | これは、結果を磁気テープ・ファイルに書き込むようにプロセスに指示します。これは常に、希望するレコードを表す文字列を取ります。こうした書込みは、Formula から返された順序で実行されます。 |
| **REPORT FILE** | これは、文字列結果を「監査」ファイルに書き込みます。 |
| **ERROR** | これは、Formula 内にエラー / 警告が検出されたことをプロセスに知らせます。プロセスは、エラーを適切に処理する必要があります。 |

## 命名規則

これらは、Formula 結果ルール表を使用する従来の方法ではインプリメントできません。次のような命名規則に準拠しています。

WRITE TO TAPE 結果には、WRITE_<result_name> という名前が付けられます。

TRANSFER 結果でも同様の規則が使用されますが、result_name 部分にはパラメータ名を使用する必要があります。たとえば、company_total_income 結果には、transfer_company_total_income という名前が付けられます。

REPORT 結果では、書込み先のファイルを示す必要があります。ファイル番号は、Formula の戻り名に含められ、たとえば次のようになります。REPORT1_<result_name> - ここでは report/audit ファイル 1 に書き込まれます。

### レポート

レポートは、磁気テープ・ファイルの作成中に記述することができます。これらのレポートを使用すれば、作成される詳細をチェックすることが可能です。1 回の実行で多数のレポートを作成できます。レポートの数は、PAY_ACTION_PARAMETERS 表の ADD_MAG_REP_FILES 処理パラメータを使用して制限できます。

各レポートへは、ファイルを示す接頭辞を使用してアクセスします。たとえば、REPORT1_ はレポート番号 1 を、REPORT2_ はレポート番号 2 をそれぞれ示します。レポート番号が ADD_MAG_REP_FILES 値の範囲外の値である場合には、無効戻りエラーが報告されます。レポート・ファイルは、必要に応じて、前述のようなファイル名でオープンされます。

### FastFormula エラー

3 つのタイプのエラーが考えられます。

- 給与計算エラー

  これらのエラーは、ERROR_PAY_<error_name> が戻されることによって識別されます。

- アサイメント・エラー

  これらのエラーは、ERROR_ASS_<error_name> によって示されます。

- 警告エラー

  これらのエラーは、ERROR_WARN_<error_name> によって示されます。

実際のメッセージ自体には、アサイメント処理 ID か給与処理 ID が接頭辞として付ける必要があります。これは、メッセージを PAY_MESSAGE_LINES 表に挿入するために行われます。警告メッセージは、アサイメント処理レベルであると見なされるため、アサイメント処理 ID を必要とします。ID が与えられないと、メッセージはログ・ファイルにのみ書き込まれます。磁気テープ・レポートの実行時には、レポートに対する処理は 1 つも存在していないので、ID を与えることはできません。処理があるのは、支払だけです。

#### 例

使用するフォーマットの例をいくつか次に示します。

| | | |
|---|---|---|
| ERROR_PAY_TEXT1 | = '50122: Unexpected value' | - 給与処理 ID 50122 と「予期せぬ値」というメッセージ |
| ERROR_PAY_TEXT1 | = 'Unexpected value' | - 給与処理 ID がなく、メッセージのみ |
| ERROR_ASS_TEXT1 | = '56988: Unexpected value' | |
| ERROR_ASS_TEXT1 | = 'Unexpected value' | |
| ERROR_WARN_TEXT1 | = '56988: Unexpected value' | |
| ERROR_WARN_TEXT1 | = 'Unexpected value' | |

## エラー処理

磁気テープは、処理を完了するか、実行全体にエラー状況をマークとして付けます。

次の 2 種類のエラーがあります。

- 致命的な給与計算処理レベルのエラー

  この種のエラーが検出された場合には、エラーが報告され、処理が終了します。

- アサイメント処理レベルのエラー

  これらのエラーは、Formula の中に設定できます。これらのエラーが検出された場合には、エラー・メッセージが報告され、処理が引き続き実行されます。次の実行までに解

決できるように、これを使用して、処理中にできる限り多くのエラーに関するレポートを作成することができます。

アサイメント処理レベルのエラーが検出された場合には、実行の終わりに給与計算処理がエラーになります。

エラー・メッセージの記述がログ・ファイルに書き込まれます。また、処理 ID がわかっている場合には、PAY_MESSAGE_LINES 表にもエントリが入れられます。

## PL/SQL 例

PL/SQL コードの次に示す部分を使用することによって、磁気テープ支払を書式化する（アサイメント処理を駆動しない）ことができます。PL/SQL プロシージャを記述するためのもう 1 つの方法は、汎用プロシージャを使用して、バッチ磁気テープ表に挿入することです。

> **注意：** この例は、'MAG Test GB' ビジネス・グループに対してのみ有効です（国別仕様 Formula は GB 専用です）。

```
create or replace package body pytstm1

  as
  CURSOR    get_assignments( p_payroll_action_id NUMBER)
  IS
  SELECT    ppp.org_payment_method_id,ppp.personal_payment_
            method_id,
            ppp.value, paa.assignment_id
  FROM      pay_assignment_actions paa, pay_pre_payments ppp
  WHERE     paa.payroll_action_id = p_payroll_action_id
  AND       ppp.pre_payment_id = paa.pre_payment_id
  ORDER BY  ppp.org_payment_method_id;

     Also need to:

     Test that the assignment are date effective?

     Order by name or person_number or other ?

     p_business_grp NUMBER;

     --

     --

     PROCEDURE new_formula
```

```
      IS
      --
      p_payroll_action_id NUMBER;
      assignment NUMBER;
      p_org_payment_method_id NUMBER;
      p_personal_payment_method_id NUMBER;
      p_value NUMBER;
      --
      --
      FUNCTION get_formula_id ( p_formula_name IN VARCHAR2)
         RETURN NUMBER IS
      p_formula_id NUMBER;
      BEGIN

                SELECT   formula_id
                INTO     p_formula_id
                FROM     ff_formulas_f
                WHERE    formula_name = p_formula_name
                AND      (business_group_id = p_business_grp
                         OR (business_group_id IS NULL
                           AND legislation_code = 'GB')
                         OR (business_group_id IS NULL AND legislation_
                         code IS NULL)
                         );
-- RETURN p_formula_id;
--
END get_formula_id;
--
BEGIN
--
pay_mag_tape.internal_prm_names(1) :=
```

```
'NO_OF_PARAMETERS'; -- Reserved positions
pay_mag_tape.internal_prm_names(2) := 'NEW_FORMULA_ID';-- --
Number of parameters may be greater than 2 because formulas
may be -- keeping running totals.--
pay_mag_tape.internal_cxt_names(1) := 'Number_of_contexts';
pay_mag_tape.internal_cxt_values(1) := 1; --
Initial value---- IF NOT get_assignments%ISOPEN THEN
-- New file-- pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_prm_values(2) := get_formula_id
('REPORT_HEADER_1');-- if
pay_mag_tape.internal_prm_names(3) = 'PAYROLL_ACTION_ID'
  then p_payroll_action_id :=
to_number(pay_mag_tape.internal_prm_values(3)); end if;--
OPEN get_assignments (p_payroll_action_id);-- ELSE----
FETCH get_assignments INTO
p_org_payment_method_id,
p_personal_payment_method_id, p_value,
assignment;-- IF get_assignments%FOUND THEN
-- New company
pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_cxt_names(2) := 'ASSIGNMENT_ID';
pay_mag_tape.internal_cxt_values(2) := assignment;
pay_mag_tape.internal_cxt_names(3) := 'DATE_EARNED';
pay_mag_tape.internal_cxt_values(3) := to_char
(sysdate,'DD-MON-YYYY');
pay_mag_tape.internal_cxt_values(1) := 3;
pay_mag_tape.internal_prm_values(2) := get_formula_id
('ENTRY _DETAIL');
ELSE-- pay_mag_tape.internal_prm_values(1) := 2;
pay_mag_tape.internal_prm_values(2) := get_formula_id
```

```
('REPORT_CONTROL_1');
CLOSE get_assignments;
-- END IF;
--END IF;--
END new_formula;
BEGIN
-- 'MAG test BG' used as an example. The business group could be
-- retrieved using the payroll action id.
 select business_group_id
 into p_business_grp
 from per_business_groups
 where name = 'MAG test BG';
--END pytstm1;
```

# 小切手ライター処理（JP 以外）

**注意：** この技術的な説明の章では、小切手ライター処理を簡単に小切手ライターと呼ぶことにします。

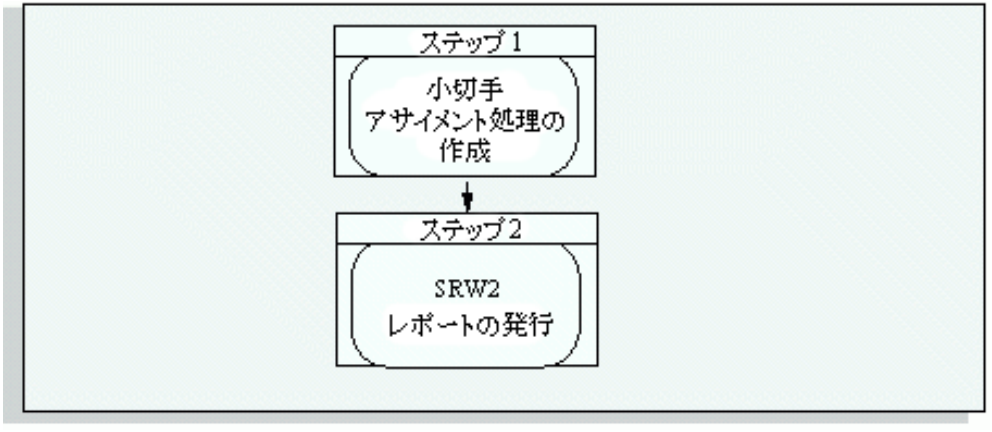
未払い支払前処理用の小切手支払を生成するには、「小切手ライター処理」を実行します。この処理を実行する前には、たとえば、SRW2 レポート、小切手の順序付けをするための「ソート順」オプションなどを必要に応じて設定しておく必要があります。

「小切手ライター」は、「標準レポートの発行（SRS)」から実行します。「磁気テープ処理」とは異なり、小切手ライターには複数のスレッドを含めることができます。

## プロセス

「小切手ライター処理」は、次の 2 つのステップから成ります。

**図 5-4 小切手ライターの各ステップ**




### ステップ 1 - 小切手アサイメント処理の作成

「小切手ライター」は、指定されたパラメータの制限に従って、ターゲット支払前処理 1 つ 1 つに対して、小切手アサイメント処理を作成します。ターゲットの支払前処理は未払いである必要があります。つまり支払が行われていないか、支払が行われても、その後無効にされている必要があります。

「小切手ライター」は、次の 2 つのステップでアサイメント処理を作成します。

1. 複数のスレッドが、以前の処理と連動するように、アサイメント処理範囲を挿入します。

   これは、「支払前処理」や「磁気テープ処理」がアサイメント処理を作成するのと同じ方法で行われます。

   参照 : プロセス : 5-19 ページ（支払前処理）

   参照 : 磁気テープ支払処理の実行 : 5-22 ページ

2. チャンクおよび小切手番号を更新するために、単一のスレッドが特定の順序ですべてのアサイメント処理で実行されます。

   順序は、カスタマイズ可能な PL/SQL プロシージャによって指定されます。スレッドは、アサイメント処理を複数のチャンクに均一に分割し、スレッドとチャンクが 1 対 1 になるようにします。さらに、各処理に小切手番号を割り当てます。

   参照 : PL/SQL プロシージャの使用および変更方法 : 5-48 ページ

この段階では、アサイメント処理のステータスは「未処理」です。

---

**注意 :** 「小切手ライター」は、アサイメントの各ターゲット支払前処理に対して、アサイメント処理と小切手を作成します。従って、1 回の「小切手ライター」の実行で、複数の小切手を単一のアサイメントに対して作成することが可能です。

---

## ステップ 2 - SRW2 レポートの発行

「小切手ライター」がアサイメント処理と連動を作成すると、各スレッドは、指定された SRW2 レポートを同期式起動コンカレント処理として発行します。レポートは、特定のチェック・フォーマットでファイルを作成します。

起動されたコンカレント処理が完了すると、アサイメント処理のステータスが「完了」に変更されます。処理が失敗した場合には、アサイメント処理のステータスが「In Error」に変更されます。このため、「小切手ライター」を再発行した場合には、レポートの発行地点から開始されます。

処理全体が完了しなければ給与計算処理が完了しないという意味では、「小切手ライター」は磁気テープ処理と似ています。しかし、「磁気テープ処理」が個別のアサイメント処理に「In Error」マークを付けることができるのに対して、「小切手ライター」はすべてのアサイメント処理に「In Error」マークを付けます。

## バッチ処理パラメータ

バッチ処理には、ユーザーが入力することのできるパラメータが多数あります。プリンタ・タイプの定義（たとえば、レポート出力用のレーザー・プリンタやライン・プリンタ）はパラメータではありません。プリンタ・タイプのデフォルトは、そのレポートに関するコンカレント処理の登録の一環として指定されます。プリンタおよびコンカレント・プログラムについての詳細は、『Oracle Applications システム管理者ガイド』をご覧ください。

**payroll_id** **オプション**

このパラメータは、アサイメントの現給与計算に従って生成される小切手を制限します。これは、ほとんどの給与計算処理の標準パラメータです。

**consolidation_set_id** **必須**

このパラメータは、「小切手ライター」のターゲット支払前処理を、その連結セットの実行に関するものに制限します。

**start_date** **オプション**

このパラメータは、「小切手ライター」がターゲット支払前処理をどれだけ前に遡って検索するか、実際に日付を指定します。このパラメータが指定されないと、「小切手ライター」は時間の初めに遡ってスキャンします。

**effective_date** **オプション**

このパラメータは、「小切手ライター」の実行の有効日を指定します。NULL の場合は、有効日は FND_SESSIONS に保持されている有効日であると見なされます。こうした行が 1 行もない場合には、SYSDATE をデフォルトとして取ります。

**payment_type_id** **必須**

このパラメータは、どの支払タイプの支払が行われるかを指定します。UK 仕様の場合は、支払カテゴリが「小切手（Cheque）」の支払タイプである必要があります。US 仕様の場合は、支払カテゴリが「小切手（Check）」の支払タイプである必要があります。

**org_payment_method_id** **オプション**

このパラメータは、ターゲット支払前処理を、その組織支払方法に関するもののみに制限します。組織支払方法で異なる小切手形式が必要とされる場合に使用されます。

**order_by_option** **必須**

このパラメータは、小切手アサイメント処理の作成および順序付けを行うためにコールする「ソート順」オプションを指定します。これをパラメータとして与えることによって、ユーザーは、生成された小切手に対して有効にする順序付け方法を指定することができます。

**report_name** **必須**

このパラメータは、「小切手ライター」が小切手の印刷ファイルやなんらかの連結給与明細などを生成するために同時に起動した SRW2 レポートの名称です。

ユーザーが拡張可能な参照が提供されています。

**start_cheque_number** **必須**

このパラメータは、生成された小切手に割り当てられる番号の連続する範囲を指定します。

**end_cheque_number** **オプション**

このパラメータは、生成された小切手に割り当てられる番号の連続する範囲を指定します。このパラメータが指定された場合、この範囲は作成される小切手アサイメント処理の数を抑制します。「小切手ライター」は、必ずしもすべてのターゲット処理を実行するとはかぎらない唯一の給与計算処理です。

終了番号が指定されないと、「小切手ライター」は、生成されたすべての小切手アサイメント処理に対して、開始番号から順次番号を割り当てて行きます。

小切手を別の連続する範囲に関して印刷しなければならない場合には（1 箱の小切手用紙の残りを、他の箱を開ける前に使い切ってしまったときに発生することがあります)、「小切手ライター」を連続する範囲ごとに個別に呼び出す必要があります。

## 小切手番号付け

詳細が印刷される小切手用紙は、通常、正式に認可されたものであり、そこには小切手番号が事前に印刷されています。UK においては、小切手の最終印刷時に番号が挿入される追加小切手番号ボックスがあるのが一般的です。給与生成システムが使用するのはこの番号です。

通常は、これら 2 つの番号は同じです。番号が一致しない場合に、なんらかのクリアリング・システムで小切手が無効とされるかどうかはわかりません。しかし、クリアリング・システムの中で小切手のパスをトレースする必要があるときには事前印刷された小切手番号が最も有用であるため、給与計算システムでは小切手支払の記録としてこの番号を記録する必要があります。

ユーザーは、「小切手ライター」によって使用されシステムで記録された小切手番号を、事前に印刷された小切手用紙の小切手番号と同じにする必要があります。場合によっては、違う番号を必要に応じて使用することも可能です。この場合、給与計算システムによって記録された小切手番号は、事前印刷の小切手番号とはまったく異なる小切手識別子になります。

**注意：** 事前印刷の用紙は、通常、1 万箱というような、まとまった単位で送られてきます。このため、給与期間の終わりに小切手を印刷発行するときに、別の範囲の小切手番号を使用する必要があるかも知れません。たとえば、1 箱の残り（9500 番～ 10000 番）を使用してから、未開封の箱（20001 番～ 30000 番）を使用して、2500 枚の小切手を印刷発行する必要がある場合もあるでしょう。「小切手ライター」は、開始小切手番号パラメータと終了小切手番号パラメータを使用して、これらの範囲の適用を強要します。

## 小切手の無効化と再発行

場合によっては小切手を無効にして、それに代わるものを任意で発行する必要があることもあります。たとえば、従業員が小切手を無くしてしまい、代わりのものを要請してきたり、その従業員は以前に退職しているので、支払をすべきでなかったということを発見したりすることがあるかも知れません。いずれの場合も、最初にすべきことは、小切手を無効にすることです。そのためには、振出口座のある銀行に連絡して、その小切手を取り消すという作業も伴います。

**注意：** 小切手を無効にしても、その支払が再度行われることが阻止されるわけではありません。

小切手を無効にして再発行するというのは、小切手をロールバックして再印刷することとは違います。小切手が実際に発行されている場合には、その小切手を無効にして、無効にした小切手を記録しておく必要があります。小切手がまだ発行されていない場合には、ロールバックします。たとえば、印刷の実行中に 1 枚の小切手でプリンタの紙詰まりが発生した場合、プリンタでは複数の小切手が印刷されたものと認識されていることがあります。こうした場合には小切手が発行されていないので、バッチ処理をロールバックして、それらの処理に対して再起動する必要があります。

無効化の理由次第では、別の小切手を発行する必要がある場合もあります。これは、「再発行」として知られています。これには追加機能は不要です。手書きの小切手を発行し、詳細をオンラインで記録するか、自動印刷にバッチ処理を再発行するかを選択することができます。

すでに支払済の処理を再処理することはできません。この処理は、支払が行われていない処理、または無効とされた処理に対してのみ支払を作成します。

## 再実行にマーク付け

「小切手ライター処理」に関しては、再実行にマークを付けることができます。ロールバック処理の場合と同様に、「小切手ライター」給与計算処理の再実行にマークを付けるときには、アサイメント・セット・パラメータを指定することによって、ユーザーがマークを付けるアサイメント処理を決定することができます。

小切手アサイメント処理の再実行にマークを付けても、それらのアサイメント処理が削除されることはなく、それらのステータスが「再実行にマーク付け」（すべての処理タイプの標準動作）に更新されるだけです。割当済の小切手番号は変更されません。このため、再実行時には、「小切手ライター」は新規印刷ファイルを作成します。

これは、小切手給与計算処理のアサイメント処理に対しては、小切手番号を再度割り当てることができないためです。給与計算処理は、指定された開始小切手番号と終了小切手番号を記録します。給与計算処理の何度かの再実行において異なる範囲の番号が使用された可能性がある場合には、給与計算処理で指定された範囲外の番号がアサイメント処理に割り当てられる可能性があります。

## 支払のロールバック

新規小切手番号を割り当てる場合には、「小切手ライター」給与計算処理とアサイメント処理をロールバックして、別個のバッチ要求を発行する必要があります。

---

**注意：** ときには、すべての小切手のロールバックを行うのが最適であることがあります。個々の小切手に再実行のマークを付けた場合は、連続した小切手番号になるとは考えにくいため、正しい、事前印刷された小切手用紙上に小切手を印刷することは難しくなります。

---

## SRW2 レポート

小切手用紙のフォーマットを設定する必要がある場合もあります。「小切手ライター」によって呼び出された SRW2 レポートには、次の 2 つのパラメータが渡されます。

- payroll_action_id（チェック処理の）
- chunk number（処理される）

このためには、このレポートは PACTID と CHNKNO という名前のパラメータを取る必要があります。

レポートが実行されるまでには、適切なアサイメント処理が作成され、パラメータによって指定された順序で小切手番号が割り当てられます。

このレポートは、指定された小切手給与計算処理およびチャンク番号のアサイメント処理を駆動しないでください。また、アサイメント処理ごとに小切手を 1 枚ずつ作成してください。小切手番号はアサイメント処理ですぐに保持されますが、支払われる金額は関連する支払前処理から取り出されます。

このレポートは、印刷時の小切手の順序を保守し、小切手番号順にアサイメント処理を実行する必要があります。

## サンプル SELECT 文

次の SELECT 文は、レポートの駆動方法を示したものです。

```
select to_number(ass.serial_number),
       ass.assignment_action_id,
       round(ppa.value,2),
       ppf.last_name,
       ppf.first_name
from per_people_f ppf,
       per_assignments_f paf,
       pay_assignment_actions ass,
       pay_pre_payments ppa
where ass.payroll_action_id =:PACTID
and ass.chunk_number =:CHNKNO
and     ppa.pre_payment_id = ass.pre_payment_id
and     ass.assignment_id = paf.assignment_id
and     ass.status <>'C'
and     paf.person_id = ppf.person_id
order by  to_number(ass.serial_number)
```

## レポートの登録

SRW2 レポートを記述したら、それを「小切手ライター」レポートとして登録する必要があります。これは、「支払前処理」の「通貨間分析ルール」の登録によく似ています。

'CHEQUE_REPORT' タイプの新規参照値も定義する必要があります。レポート名と摘要を入力します。

「磁気テープ処理」と同様の方法で、レポートによって生成されたファイルに名前が付けられます。

*p<trunc(conc_request_id,5)>.c<chunk_number>*

要求 ID の長さが 5 文字より短い場合には、ファイル名は 0（ゼロ）で埋められます（例 :p03451.cl）。

$ APPLCSF が定義されているときには $APPLCSF/$APPLOUT ディレクトリに書き込まれ、それ以外の場合は $PAY_TOP/$APPLOUT に書き込まれます。

「小切手ライター」は、複数のスレッドによって実行された場合には、いくつかのファイルを作成します。これは、スレッドとチャンクが 1 対 1 になるように、「小切手ライター」アサイメント処理がいくつかのチャンクに分割されるためです。このため、各スレッドは、チャンクを取り出して、処理することができます。これは、複数のプロセッサが搭載されたマシンのパフォーマンスの向上を目的としています。たとえば、処理するスレッドが 4 つあれば、作成されるファイルは 4 つになります。

- p03451.c1
- p03451.c2
- p03451.c3
- p03451.c4

小切手ライターは 5 番目のファイルを作成します（4 つのファイルを 1 つに連結する処理により）。このファイルの名前は p03451.ch となります。

## PL/SQL プロシージャの使用および変更方法

PL/SQL プロシージャの使用 / 変更方法 「小切手ライター」は、PL/SQL プロシージャによって決定され、このプロセスから匿名でコールされた順序の小切手およびチャンク番号を持つアサイメント処理を更新します。汎用製品に関しては、デフォルトの PL/SQL プロシージャ、pay_chqwrt_pkg.chqsql が提供されています。

デフォルトのソート順は次のとおりです。

1. 組織
2. 部門
3. 姓
4. 名

このプロシージャを変更して、プロシージャに渡されたフラグによって示された基準に従って、数種類のソート順を設定することも可能です。ご自分のビジネス・ニーズにあわせて順序付けするには、コア SELECT 文をコピーし、副問合せを変更します。

SQL 文全体へのアクセスを可能にすることの利点としては、小切手を任意の基準に従って順序付けできるということがあげられます。ORDER BY 句の指定のみを認めていたなら、順序付けはコア SQL 文の FROM 句にすでに存在している表の属性だけに限られていたことでしょう。

新たに必須要件でのソートを設定するには、pay_chqwrt_pkg.chqsql パッケージ・プロシージャを変更します。procname 変数を確認するときに、次の IF 文を追加してください。

```
else if procname = 'NEW ORDER BY' then

   sqlstr := 'select ....'
```

SELECT 文には既存の SELECT 文をコピーして使用することも可能ですが、「句でソート」は変更してください。SELECT 文は、アサイメント処理の ROWID を返す必要があります。

こうした情報に基づいて、アサイメント処理にシリアル / 小切手番号を与え、チャンクに割り当てることができます。

同様に、SRW2 レポートの場合と同じように、オプションでのソートを新たに使用するためには、その前に登録が必要です。これは、参照タイプが CHEQUE PROCEDURE であることを除いては、同様の方法で行われます。内容フィールドには意味のある説明を、摘要フィールドにはオプション名（たとえば、NEW ORDER BY）をそれぞれ入力してください。

# 現金処理

「現金処理」は、支払が行われたことをシステムに知らせ、支払前処理がロールバックされないようにします。

**注意：** これは UK、JP 専用の処理です。

# 給与計算処理パラメータ

給与計算処理パラメータは、Oracle Payroll バッチ処理の様々な局面を制御するシステム・レベルのパラメータです。特定のパラメータに対して値を設定した場合には、システム全体にその影響が及ぶ可能性があるということを認識しておくことが大切です。本文では、パラメータが特定の処理に関連しているということについて説明します。一部のパラメータに関しては、配列処理の概念やパフォーマンスに与える影響についても理解しておく必要があります。

## 処理パラメータ値

各パラメータの値はシステム定義されていますが、初期導入プロセスの一環として、また、パフォーマンスを調整するために、これらの値を上書きすることも可能です。

処理パラメータは、次の 2 つの列からなる下表 : **PAY_ACTION_PARAMETERS** 表に適切な行を挿入することによって指定されます。

```
PARAMETER_NAME NOT NULL VARCHAR2(30)
PARAMETER_VALUE NOT NULL VARCHAR2(80)
```

給与計算バッチ処理では、起動時にこの表から値が読み込まれるか、特定のパラメータ値が指定されない場合には、適切なデフォルトが提供されます。

## 処理パラメータの概要

ユーザー入力可能な処理パラメータとそれらの値を事前定義のデフォルト値とともに次にリストします。

**注意 :** これらのパラメータに関しては、大文字 / 小文字が重要な意味を持ちます。

| パラメータ | 値 | デフォルト |
|---|---|---|
| ADD_MAG_REP_FILES | 1 以上 | 4 |
| BAL BUFFER SIZE | 1 以上 | 30 |
| CHUNK SHUFFLE | Y または N | N |
| CHUNK_SIZE | 1 - 16000 | 20 |
| EE BUFFER SIZE | 1 以上 | 40 |
| LOG_AREA | 次を参照 | |
| LOG_ASSIGN_END | 次を参照 | |
| LOG_ASSIGN_START | 次を参照 | |

| | | |
|---|---|---|
| LOGGING | 次を参照 | |
| MAX_ERRORS_ALLOWED | 1 以上 | CHUNK_SIZE または 20（チャンク・サイズがない場合） |
| MAX_SINGLE_UNDO | 1 以上 | 50 |
| RR BUFFER SIZE | 1 以上 | 20 |
| RRV BUFFER SIZE | 1 以上 | 30 |
| COST BUFFER | 1 以上 | 20 |
| THREADS | 1 以上 | 1 |
| TRACE | Y または N | N |
| USER_MESSAGING | Y または N | N |

**注意：** 下線のないパラメータ名のすべてに、下線付きのパラメータ名があります（CHUNK SHUFFLE を除く）。

# パラレル処理パラメータ

### THREADS

```
Parameter Name:THREADS
Parameter Value: 1 以上
Default Value:1
```

Oracle Payroll は、マルチプロセッサ・マシンを活用するように設計されています。このため、処理を多数の「スレッド」に分割することにより、バッチ処理のパフォーマンスの向上を図ることができます。こうしたスレッド、すなわちサブプロセスは並行して実行されます。

コンカレント・マネージャに対してバッチ処理を発行した場合、THREADS パラメータにより、コンカレント・マネージャのもとで実行されるサブプロセスの総数が決定します。マスター・プロセスは、（THREADS-1）サブプロセスを発行します。

このパラメータをサーバー側のパフォーマンスを最適化する値に設定してください。単一プロセッサ・マシンの場合は、デフォルト値 1 が設定されます。マルチプロセッサ・マシンのベンチマーク・テストによると、最適値は 1 台のプロセッサ当たり約 2 つのプロセスです。したがって、たとえばサーバーが 6 台のプロセッサを備えている場合には、初期値を 12 に設定して、この値によるバリエーションのパフォーマンスへの影響をテストしてください。

**注意：** 必要数のサブプロセスを並行して実行できるように、コンカレント・マネージャを定義する必要があります。これは、アプリケーションのシステム管理者が行ってください。

### CHUNK_SIZE

```
Parameter Name:CHUNK_SIZE
Parameter Value: 1 - 16000
Default Value: 20
```

バッチ処理の各コミット単位のサイズ。このパラメータは、処理の初期段階において挿入されるアサイメント処理の数と、メイン処理段階において一度に処理されるアサイメント処理の数を決定します。

**注意：** これは、「小切手ライター」、「磁気テープ」、「遡及計算」の各処理には当てはまりません。

処理の初期段階においては、このパラメータは挿入用の配列サイズを定義します。大きなチャンク・サイズ値は望ましくありません。ベンチマーク・テストの結果としてデフォルト値が設定されています。

各スレッドは、一度に 1 つのチャンクを処理します。

## 配列の選択、更新および挿入用のバッファ・サイズ・パラメータ

次に示すパラメータは、「メモリー内」配列処理に使用されるバッファ・サイズを制御します。この値によって、バッファに保持できる行数が決まります。

**注意：** これらのパラメータは、「給与計算処理」にのみ適用されます。

これらのパラメータの値を設定する場合には、配列サイズ、パフォーマンス、メモリーの各要件を考慮する必要があるということに注意してください。一般に、一度にフェッチ、更新または挿入される行数が多ければ多いほど、パフォーマンスは向上します。しかし、20 行を超えたあたりで向上幅は低下します。

したがって、1 ～ 20 までの値のパフォーマンスの向上が著しいのに対して、20 ～ 100 までの値のパフォーマンスの向上はほとんど見られません。値が大きいほどメモリーの使用量も多いということにも注意してください。このような理由から、デフォルト値を変更しても何の利点もないと思われます。

### CHUNK_SIZE

```
Parameter Name: CHUNK_SIZE
Parameter Value: 1 - 16000
Default Value: 20
```

バッチ処理の各コミット単位のサイズ。前と同様。

### RR BUFFER SIZE

```
Parameter Name:RR BUFFER SIZE
Parameter Value: 1以上
Default Value: 20
```

配列の挿入および更新に使用される「実行結果」バッファのサイズ。1 行 / 実行結果。

### RRV BUFFER SIZE

```
Parameter Name:RRV BUFFER SIZE
Parameter Value: 1以上
Default Value: 30
```

配列の挿入および更新に使用される「実行結果値」バッファのサイズ。1 行 / 実行結果値。通常は、(RR BUFFER SIZE * 1.5) に設定されます。

### BAL BUFFER SIZE

```
Parameter Name:BAL BUFFER SIZE
Parameter Value: 1以上
Default Value: 30
```

配列の挿入および更新に使用される「最新バランス」バッファのサイズ。1 行 / 最新バランス。

### EE BUFFER SIZE

```
Parameter Name:EE BUFFER SIZE
Parameter Value: 1以上
Default Value: 40
```

1 つのアサイメント当たりの「エレメント・エントリ」、「エレメント・エントリ値」、「実行結果」、「実行結果値」の初期配列選択に使用されるバッファのサイズ。

## 原価計算固有のパラメータ

### COST BUFFER SIZE

Parameter Name:COST BUFFER SIZE
Parameter Value: 1以上
Default Value: 20

原価計算処理で配列の挿入および選択に使用されるバッファのサイズ。

## 磁気テープ固有のパラメータ

### ADD_MAG_REP_FILES

Parameter Name:ADD_MAG_REP_FILES
Parameter Value: 1以上
Default Value: 4

磁気テープ処理で作成できる追加監査 / レポート・ファイルの最大数。

## エラー・レポート・パラメータ

すべての支払サイクルにおいて、なんらかのエラーが個別のアサイメントの処理、特に「給与計算」で発生すると思われるでしょう。こうしたエラーは、通常、従業員レコード内の不正確なデータや欠落データに起因します。実際には、単一のアサイメントの失敗で実行全体が失敗に終わることは本意ではないと思われます。しかし、多くのアサイメントが次々にエラー条件を引き起こした場合、通常は深刻な問題であるため、処理全体を停止して原因を調査する必要があります。アサイメント・レベルのエラーをサポートする処理に関しては、MAX_ERRORS_ALLOWED パラメータを使用して、こうしたエラーを調査するために処理全体を停止するポイントを制御できます。

この機能を使用する処理は次のとおりです。

- 給与計算
- 支払前処理
- 原価計算
- ロールバック

#### MAX_ERRORS_ALLOWED

Parameter Name:MAX_ERRORS_ALLOWED
Parameter Value: 1以上
Default Value: CHUNK_SIZEまたは20 （チャンク・サイズがない場合）

処理全体のステータスが「エラー」にされる前にエラーとなる可能性のある連続する処理の数。

## ロールバック固有のパラメータ

特定の給与計算処理のロールバックは、次の2つの方法で実行できます。「要求の発行」ウィンドウからバッチ処理を発行することができます。別の方法として、「給与計算処理結果」または「アサイメント処理結果」ウィンドウから特定の処理を削除することによって、その処理をロールバックすることも可能です。ウィンドウからロールバックした場合には、このパラメータはコミット単位サイズを制御します。

#### MAX_SINGLE_UNDO

Parameter Name:MAX_SINGLE_UNDO
Parameter Value: 1以上
Default Value: 50

ロールバックがフォームから実行された場合に単一のコミット単位でロールバックすることのできるアサイメント処理の最大数。デフォルト制限を変更することも可能ですが、通常は、デフォルト制限を超える可能性がある場合には、SRS画面からロールバック処理を使用します。

## 給与計算処理のログ

Oracle Payrollシステムのインストールおよびテスト中には、本製品で提供されている詳細ロギング・オプションを必要に応じて「オン」にすることができます。**LOGGING**パラメータを使用して、問題の調査に有用な大量の詳細情報を提供してください。

詳細ロギング・オプションは、他の方法では識別し難い問題を調査する必要がある場合にのみ「オン」にしてください。ロギング・アクティビティは、ログしている処理の全体的なパフォーマンスに影響を及ぼします。一般的には、この機能は、実際に作業に取り掛かる前の初期導入ならびにテスト段階において必要とされます。通常の操作では、詳細ロギングを「オフ」にしておいてください。

---

**注意：** 給与計算処理の実行中に発生する問題の識別方法や解決方法についてOracleサポートにお問合せになる場合には、最初にログ・ファイルをご用意ください。ロギング・カテゴリ、エリア、アサイメント範囲を定義してから、問題の処理を再発行してください。

---

## ロギング・カテゴリ

ロギング・カテゴリは、ログに含まれる情報のタイプを定義します。これによって、問題の原因であると思われる特定のエリアにのみ注意を注ぐことができます。こうした情報は、次のような複数の値を指定することによって、任意の数のみ設定できます。

- **G**　一般的な（特定のカテゴリでない）ロギング情報

  一般的な情報に関する PY_LOG マクロからの出力メッセージ。このオプションは出力をソートしないので、通常はユーザーが特定のカテゴリのリストを選択します。

- **M**　入力または終了ルーティング情報

  'In: pyippee', 'Out: pyippee' といったメッセージによって、なんらかの関数が入力および終了されたことを示す出力情報。この情報は、コール・レベルを示すことを目的とし、ファンクション・コール・レベルのコードが通った経路をトレースするのに使用することができます。多くの場合、これはコア・ダンプなどの問題を突き止めようという場合に有用です。

- **P**　パフォーマンス情報

  アサイメント・レベルや実行レベルにおける特定の操作の発生回数と、その操作が発生した理由を示す出力情報。たとえば、バランス・バッファ配列の記述。

- **E**　エレメント・エントリ情報

  アサイメントのエントリが取り出された後に、その構造のなんらかの項目（たとえば、間接要素の追加または更新）が変更された場合の、メモリー内エレメント・エントリ構造のステータスを示す出力情報。エントリの処理も示します。

- **L**　バランス取出し情報

  最新バランスの取出しおよび後続の失効段階を示す出力情報。

- **B** バランス保守情報

  メモリー内バランスの作成および保守を示す出力情報。

- **I**　バランス出力情報

  バランス・バッファからデータベースに書き込まれた値の詳細を示す出力情報。

- **R**　実行結果情報

  「実行結果」または「実行結果値」バッファからデータベースに書き込まれた実行結果および実行結果値の詳細を示す出力情報。

- **F**　Formula 情報

  Formula の実行の詳細を示す出力情報。これには、Formula のコンテキスト、入力および出力が含まれます。

- **C**　C キャッシュ構造情報

  給与計算キャッシュ構造と構造内のエントリに対する変更の詳細を示す出力情報。

- **Q**　C キャッシュ問合せ情報

  給与計算キャッシュ構造で実行されている問合せを示す出力情報。

- **S**　C キャッシュ終了ステータス情報

  成功で終了するかエラーで終了するか、処理の終了前の給与計算キャッシュの状態を示す出力情報。ロギング情報の多くには ID 値が含まれるため、ローカル・データベースへのアクセスが不可能な場合に、これを使用して相互参照を与えることができます。

- **V**　Vertex（US およびカナダの顧客のみが使用可能）

  Vertex 税金エンジンとの間で受け渡される値を示す出力情報。

  このオプションは、エンジンの内部設定を示す Out ディレクトリに別個のファイルも作成します。

# ロギング・パラメータ

### LOGGING

```
Parameter Name:LOGGING
Parameter Value: G, M, P, E, L, B, I, R, F, C, Q, S, V
Default Value:No logging
```

### LOG_AREA

```
Parameter Name: LOG_AREA
Parameter Value:Function to start logging
Default Value:No default
```

### LOG_ASSIGN_START

```
Parameter Name:LOG_ASSIGN_START
Parameter Value:Assignment to start logging
Default Value:All assignments
```

### LOG_ASSIGN_END

```
Parameter Name:LOG_ASSIGN_END
Parameter Value:Assignment to end logging, including this one
Default Value:All assignments
```

## 出力ログ・ファイル

ロギング・オプションを使用可能にした場合には、出力がコンカレント・マネージャによって作成されたログ・ファイルに自動的に含まれます。このログ・ファイルの内容をレビューまたは印刷することが可能です。

「一般」カテゴリは別として、ログ・ファイルには ID 値を使用した簡潔な形式の情報が含まれます。このため、ログ・ファイルの大きさが最小限に保たれる一方で、ユーザーが必要とするすべての技術詳細が提供されます。

'G' と 'M' 以外の各ロギング・カテゴリの出力を理解しやすくするために、ログ・ファイルには、正確なフォーマットを示すヘッダーが含まれています。

## その他のパラメータ

### USER_MESSAGING

```
Parameter Name:USER_MESSAGING
Parameter Value:Y/N
Default Value:N
```

ユーザー可読情報の pay_message_lines 表への詳細ロギングを使用可能にするには、このパラメータを 'Y' に設定します。この情報には、「給与計算」中に処理される上書きやエレメントに関する詳細が含まれます。

**注意：** この情報は問題を調査する場合には有用ですが、通常の作業には詳細すぎるかもしれません。

### TRACE

```
Parameter Name:TRACE
Parameter Value:Y/N
Default Value:N
```

データベース・トレース機能を使用可能にするには、このパラメータを 'Y' に設定します。Oracle トレース・ファイルが生成され、ご使用のプラットフォーム用の標準出力ディレクトリに保存されます。

**警告：** トレース機能は、問題を調査する場合にのみ使用してください。この値を 'Y' に設定すると、データベースのパフォーマンスに著しい低下が生じる恐れがあります。給与計算処理のパフォーマンスに重大な問題が生じた場合には、このパラメータがデフォルト値 'N' に再設定されているか確認してください。

## QuickPay 処理のシステム管理

QuickPay 実行処理または QuickPay 支払前処理を開始した場合、画面は処理が完了するまで停止します。QuickPay は、コンカレント・マネージャが指定された期間内に処理を開始できなかった場合や、処理を開始はしたものの、指定された期間内に完了できなかった場合など、あらゆるケースを管理するように設定されています。こうした状況は、多数の優先順位の高い処理が一度にコンカレント・マネージャにアクセスした場合などに時として発生する可能性があります。

QuickPay 処理を開始した場合に発生する画面停止のシステム管理には、次が含まれます。

- 処理を完了させるために、2 ～ 3 秒ごとにコンカレント・マネージャをチェックする。
- 最大待機時間内に処理が完了しなかった場合に、画面の停止を解除し、ユーザーにエラー・メッセージを送信する。

  エラー・メッセージには、処理の AOL コンカレント要求 ID が含まれます。ユーザーは、処理を再問合せして、処理の現状を確認する必要があります。

システム管理者は、次のことを行うことによって、インストール時に QuickPay の処理速度を高めることができます。

- 処理完了の確認が行われる間隔のデフォルトを変更する。

  デフォルトでは、コンカレント・マネージャのチェックは、2 秒間隔で行われます。PAY_ACTION_PARAMETERS 表の QUICKPAY_INTERVAL_WAIT_SEC パラメータ行によってこのデフォルトが設定されます。

- 最大待機時間のデフォルトを変更する。

  QuickPay 処理の最大待機時間のデフォルトは、システムがエラー・メッセージを発行してから 300 秒（5 分）です。PAY_ACTION_PARAMETERS 表の QUICKPAY_MAX_WAIT_SEC パラメータ行によってこのデフォルトが設定されます。

- QuickPay の実行処理と支払前処理専用の新規コンカレント・マネージャを定義する。

### チェックが行われる間隔または最大待機時間のデフォルトを変更する場合：

❑ PAY_ACTION_PARAMETERS 表に新規行を挿入するか、既存の行を更新します。

QUICKPAY_INTERVAL_WAIT_SEC および QUICKPAY_MAX_WAIT_SEC は、参照タイプ ACTION_PARAMETER_TYPE 用のコードであることに注意してください。

### 2 種類の QuickPay 処理専用の新規コンカレント・マネージャを定義する場合 :

1. 標準コンカレント・マネージャの専用化ルールから 2 種類の QuickPay 処理を削除します。
2. 削除した処理を、標準コンカレント・マネージャの専用化ルールより数の少ない新規 QuickPay コンカレント・マネージャの専用化ルールに含めます。そうすることにより、QuickPay 処理要求の開始に要する時間が削減されます。

# アサイメント・レベル連動

給与計算処理を行うときに、一連の処理を実行します。各処理は、1 つのアサイメント処理を実行します。

処理の実行順序は、処理の成功にきわめて重要です。これは、各処理が実行順序の前の処理の結果を使用するだけでなく、その結果を基に成り立っているためです。処理する順序もまた、データの整合性の問題によって決定されます。たとえば、支払前処理（支払方法に応じて支払を準備する）では給与計算処理（総支給額と差引支給額を計算する）の結果が使用されます。

正しく支払うためには、支払前結果も変更しなければ結果を変更することはできないということがきわめて重要となります。このような事態が発生しないようにするため、さらにはデータの整合性を実現するため、Oracle Payroll ではアサイメント・レベルでの連動ルールが採用されています。

## 処理分類

給与計算処理（給与計算や原価計算など）および処理タイプ（QuickPay など）は、「順序指定」または「順序未指定」として分類されます。処理分類によって、連動処理ルールの適用方法が決まります。

| 処理および処理タイプ | 分類 | 連動行の挿入 |
|---|---|---|
| 給与計算 | 順序指定 | No |
| QuickPay | 順序指定 | No |
| リバーサル | 順序指定 | Yes |
| バランス調整 | 順序指定 | No |
| バランス初期化 | 順序指定 | No |
| 支払前処理 | 順序未指定 | Yes |
| QP 支払前処理 | 順序未指定 | Yes |
| 現金支払 | 順序未指定 | Yes |
| 磁気テープ転送 | 順序未指定 | Yes |
| 前払 | 順序指定 | No |
| 小切手ライター | 順序未指定 | Yes |
| 現金 | 順序未指定 | Yes |
| 原価計算 | 順序未指定 | Yes |
| GL への転送処理 | 順序未指定 | Yes |
| 差額遡及 | 順序指定 | No |
| レトロペイ | 順序指定 | No |

## 順序指定処理

これらの処理は、同じレベルに存在しているため、厳密な順序で処理する必要があります(たとえば、QuickPay 前の給与計算処理など)。将来において別の順序指定処理が存在している場合、もしくは過去において不完全な順序指定処理が存在している場合には、そのアサイメントに関する順序指定処理を挿入することはできない、というのが一般的なルールです。

ただし、「リバーサル」と「バランス調整」に関しては例外です。さらに、このルールに影響を及ぼす特定の国別仕様要件がある場合もあります。詳細は、5-62 ページの「給与期間依存国別仕様」を参照してください。

順序ルールには、給与計算処理の有効日が用いられます。同じ有効日を持つ処理が 2 つ以上存在する場合には、処理順序番号によって処理する順序が決定されます。

## 順序未指定処理

将来または過去において、アサイメント処理が他にも存在しているとしても、そのアサイメントに関する順序未指定処理を挿入することができます。たとえば、「原価計算処理」は、「支払前処理」を実行する前と後のいずれでも実行できます。

## 給与期間依存国別仕様

複数のアサイメントを持つ従業員の税計算を管理するルールは、国別仕様によって異なります。また、このルールによって連動ルールの適用方法が決まります。

たとえば、UK においては、税金を計算するときには、給与期間内のあらゆるアサイメントの全支給項目を考慮に入れる必要があります。この種の国別仕様に関しては、連動ルールによってすべてのアサイメントの処理順序がチェックされるため、別のアサイメントに適用される処理によって、給与期間内の 1 つのアサイメントに障害が引き起こされる可能性があります。

たとえば、月次と週次の両方で給与計算される従業員を処理する場合に、週次の給与計算に基づいて処理し、支払を行った後で、その従業員の月次での支払の実行をロールバックすることはできません。週次アサイメントの支払処理をロールバックしなければ、月次の給与計算処理をロールバックすることはできません。

US などの国別仕様においては、各アサイメントが個別に処理されるため、1 つのアサイメントで連動障害が発生しても、他のアサイメントに障害が発生することはありません。

## 処理連動行

アサイメント処理に対して連動が挿入されると、処理中のアサイメント処理がロックされます。たとえば、支払前処理連動は、支払が行われる給与計算処理を指すため、これにより給与計算が削除されないようにロックされます。順序指定処理が存在する場合には、その処理の前に、順序指定処理が挿入されることはありません。順序指定処理は順序どおりに発生する必要があります。

## 「再実行にマーク付け」処理のチェック

再実行にマークが付けられたアサイメント処理に関しては、特別なルールが 1 つあります。「給与計算」または「QuickPay」処理を再実行しようとすると、処理しようとしているアサイメント（または、国別仕様によっては個人）に関して、再実行にマーク付けされた順序指定アサイメント処理が過去に存在していないかどうかが、システムによりチェックされます。

## 順序指定処理の特別ルール

次の状況のいずれかが存在している場合には、アサイメント処理は挿入されません。

- 挿入日と同じかそれ以前の日付を持つアサイメントに関して、不完全な順序指定処理がある。
- 挿入日以降の日付で、なんらかの処理ステータスを持つアサイメントに関して、順序指定処理がある。
- 挿入日以降の日付で、削除不可能処理がある。

ただし、次の 2 つは例外です。

- リバーサル
- バランス調整

リバーサルまたはバランス調整が挿入されると、システムは、その給与期間の後の方に存在しているアサイメント処理の処理順序番号を変更することによってその処理順序を保守します。

## 順序未指定処理の特別ルール

別の順序未指定アサイメント処理から現在処理されているアサイメント処理に関して連動が存在している場合には、この順序未指定アサイメント処理は挿入されません。

たとえば、QuickPay に続いて QuickPay 支払前処理を実行した場合には、後続の支払前処理によってアサイメント処理 / 連動が QuickPay に挿入されることはありません。これは、QuickPay 支払前処理がすでに処理と連動を挿入していると考えられる上に、支払前処理が同じ処理分類を持つためです。

## ロールバック・ルールと再実行にマーク付けルール

この表は、給与計算処理とアサイメント処理の再実行およびロールバックに関するルールを要約したものです。処理によっては、個々のアサインメントの処理のみをロールバックすることはできないものがあります。たとえば、磁気転送処理の場合に個別のロールバックは実行できません。実際には、この処理では磁気テープ・ファイルが作成されるため、処理全体をロールバックした後で再実行する必要があります。

| | 給与処理 | | アサイメント処理 | |
|---|---|---|---|---|
| **処理タイプ名** | **再実行** | **ロールバック** | **再実行** | **ロールバック** |
| 給与計算 | Yes | Yes | Yes | Yes |
| QuickPay | Yes | Yes | Yes | No |
| リバーサル | No | Yes | No | No |
| バランス調整 | No | Yes | No | No |
| バランス初期化 | No | Yes | No | No |
| パージ | Yes | No | No | No |
| 支払前処理 | Yes | Yes | Yes | Yes |
| QP 支払前処理 | Yes | Yes | Yes | No |
| 現金支払 | No | Yes | No | No |
| 磁気テープ転送 | Yes | Yes | No | Yes |
| 小切手ライター | Yes | Yes | Yes | Yes |
| 現金 | No | Yes | No | Yes |
| 原価計算 | Yes | Yes | Yes | Yes |
| GL への転送処理 | Yes | Yes | No | No |
| 前払 | Yes | Yes | Yes | Yes |
| レトロペイ | Yes | Yes | Yes | Yes |
| 差額遡及 | Yes | Yes | Yes | Yes |

### 順序指定処理のロールバック

そのアサイメントに関するその後の順序指定処理が存在している場合には、順序指定処理をロールバックすることはできません。ただし、バランス調整やリバーサルは別です。たとえば、次の給与期間で別の給与計算がすでに実行されている場合には、その給与期間の給与計算をロールバックすることはできません。

## 処理の再実行にマーク付け

そのアサイメントに関するその後の順序指定処理が存在している場合には、バランス調整やリバーサルは別として、順序指定処理の再実行にマークを付けることはできません。ただし、ロックの原因となる先日付の処理自体の再実行にマークが付けられている場合には可能です。

ロック処理自体の再実行にマークが付けられている場合には、順序未指定処理を再実行することができます。

# GL への転送処理

（原価計算結果を累計する）Post-Run 処理「原価計算」を実行したら、いつでも実行結果を Oracle General Ledger やその他のシステムに転送できます。

この処理は、給与計算と同じ方法で、複数のスレッドを使用して発行できます。

# 原価計算処理

給与計算処理を実行したら、Post-Run 処理である「原価計算」を開始します。「原価計算処理」は、GL への転送を初めとするアプリケーションの実行結果を累計します。この処理は、あらゆるレベルの「原価割当フレックスフィールド」からユーザーが選択した情報に従って、実行結果を次によってソートします。

- 会社
- 会計帳簿
- 原価センター
- GL
- 労務の分配勘定科目

給与計算結果の原価割当例と選択された従業員支給項目への事業主費用の分配例が、次表に示されています。

Oracle General Ledger もインストールされている場合には、「原価計算処理」を実行した後に、「GL への転送処理」を実行してください。これによって、「原価計算処理」から Oracle General Ledger に実行結果が転送されます。

## 給与の原価割当例

次表には、4 人の従業員の給与計算実行結果が、「原価割当」キー・フレックスフィールドを使用して指定された勘定科目とワーク・ストラクチャを使用して示されています。「原価計算処理結果表」の例は、「原価計算処理」がこれらの給与計算結果を次に対してどのように割り当てるかを示したものです。

- GL の勘定科目および原価センター
- 原価センターの勘定科目および原価センター内の製品ライン（労務の分配用）

**給与計算処理結果のサンプル**

| 従業員 | ワーク・ストラクチャ | | 支給項目と控除項目 | | | |
|---|---|---|---|---|---|---|
| | 原価センター | 製品ライン | 給与 | 賃金 | 残業 | 組合費 |
| 従業員 1 | 製造 | H201 100% | | 1,000 | 400 | 20 |
| 従業員 2 | 販売 | H305 100% | 1,500 | | | |
| 従業員 3 | 製造 | H201 50%<br>H202 50% | | 2,000 | 600 | 30 |
| 従業員 4 | 販売 | H305 20%<br>H310 40% | 1,000 | | | |

次の表は、前記のサンプル実行結果に基づいた原価割当を示したものです。

**原価計算プロセス結果の例**

| 勘定科目コード | 原価センター | | 製品ライン | | | | |
|---|---|---|---|---|---|---|---|
| | 製造 | 販売 | H201 | H202 | H305 | H307 | H310 |
| 給与 | | 2,500 | | | 1,700 | 400 | E400 |
| 賃金 | 3,000 | | 2,000 | 1,000 | | | |
| 残業 | 1,000 | | 700 | 300 | | | |
| 組合費 | 50 | | | | | | |
| 精算 | 勘定には、支給項目である給与、賃金および残業という貸方バランス、そして控除項目である組合費という借方バランスが含まれます。 | | | | | | |

# 事業主費用の分配例

事業主費用を表すエレメントと原価タイプ「分配」をリンクすると、「原価計算処理」は事業主費用を各従業員の間接費として一連の従業員の支給項目に分配します。この例は、総額100ドルの事業主支払が賃金や残業などの一連の支給項目にどのように分配されるかを、原価センター「製造」と製品ライン H201 および H202 に関して示したものです。

## 「製造」原価センターの間接費分配

```
Total paid to Production Cost Center as Wages run result: $3,000.00

Total paid to Production Cost Center as Overtime run result:
$1,000.00

Total for Earnings types specified for Distribution: $4,000.00

Ratio for Wages distribution, Production Cost Center = 3000/4000 =
.75

Wages overhead = Pension Charge 100 x .75 = 75.00

Ratio for Overtime distribution, Production Cost Center = 1000/4000
= .25

Overtime overhead = Pension Charge 100 x .25 = 25.00
```

## 製品種目 H210 および H202 の間接費分配

```
Total paid for Product Line H201 as Wages run result: $2,000.00

Total paid for Product Line H202 as Wages run result: $1,000.00

Total paid for Product Lines H201 and H202 as Wages: $3,000.00


Ratio for Wages distribution, Product Line H201 = 2000/3000 = 0.6667
```

```
Product Line H201 overhead = Total Wages overhead $75 x .6667 =
$50.00

Ratio for Wages distribution, Product Line H202 = 1000/3000 = 0.3334

Product Line H202 overhead = Total Wages overhead $75 x .3334 =
$25.00

Total paid for Product Line H201 as Overtime run result: $700.00

Total paid for Product Line H202 as Overtime run result: $300.00

Total paid for Product Lines H201 and H202 as Overtime: $1,000.00


Ratio for Overhead distribution, Product Line H201 = 700/1000 = .7

Product Line H201 overhead = Total Overtime overhead $25 x .7 =
$17.50

Ratio for Overhead distribution, Product Line H202 = 300/1000 = 0.3

Product Line H202 overhead = Total Overtime overhead $25 x .3 =
$7.50
```

**原価センターと製造ラインの間接費分配の合計**

| 勘定科目コード | 原価センター | 製品ライン | |
|---|---|---|---|
| | 製造 | H201 | H202 |
| 賃金 | 3,000 | 2,000 | 1,000 |
| 事業主分配金 | 75 | 50 | 25 |
| 残業 | 1,000 | 700 | 300 |
| 事業主分配金 | 25 | 17.50 | 7.50 |

# 6

# 給与アーカイブ・レポータ（PAR）処理

## 給与アーカイブ・レポータ（PAR）処理

給与アーカイブ・レポート（PAR）処理を使用すると、税務年度の終わりや各税務四半期など定期的な周期で、従業員アサイメントの複雑な給与計算レポートを作成できます。これらのレポートは磁気テープを使用して、税務署やその他の政府機関に提出できます。

必要であれば、レポート上のデータをそのまま正確にアーカイブすることもできます。これにより、給与担当部門、またはレポートを受け取る外部機関で、将来このデータを検討する必要が生じた場合にも対応できます。

アーカイブが必要ない場合でも、レポートが作成されたこと、そしてどのような従業員アサイメントがレポートに含まれていたかを記録することができます。

PAR 処理は主に磁気テープ・レポートで使用されますが、Oracle Report Writer を使用して提供されるレポートでも使用することができます（「アーカイブ」モードで）。

ここで説明する汎用の PAR 処理は、人事管理システムの給与計算のすべての国別仕様の要件を満たすものではない場合があります。このため、給与計算処理、給与または組織を含むようにレポート対象のデータが拡大されるような変更がローカライズ・チームによって行われている場合があります。

### PAR のモード

柔軟な使用を可能にするために、PAR は 3 つの異なるモードで実行できます。

- アーカイブ付き磁気テープ

  このモードでは、PAR は FastFormula アーカイブ表を使用してレポートに必要な値をアーカイブします（FF_ARCHIVE_ITEMS および FF_ARCHIVE_ITEM_CONTEXTS）。その後、アーカイブされた値に基づいて、磁気テープ上にレポートを作成します。

- アーカイブ

  このモードでは、PAR は FastFormula アーカイブ表を使用してレポートに必要な値のアーカイブのみを行います。

PAR 処理を「アーカイブ」モードで実行することにより、Oracle Report Writer または磁気テープ処理を使用して、FastFormula アーカイブ表からデータを抽出できます。

- アーカイブなし磁気テープ

  このモードでは、PAR は磁気テープ上にレポートを作成し、レポート作成の記録（PAY_PAYROLL_ACTIONS 表の中に）、またはレポートされている個別のアサイメントの記録（PAY_ASSIGNMENT_ACTIONS 表の中に）、あるいはその両方を保存します。

  **注意：** 代替の PYUMAG プロセスを使用して磁気テープ・レポートを作成した場合は、レポート作成の記録は残りません。

それぞれが大量のデータのサブセットを必要とする多数のレポートを磁気テープにより作成する必要がある場合には、PAR を「アーカイブ」モードで実行した後で、「アーカイブなし磁気テープ・モード」で実行すると便利です。すべてのデータを「アーカイブ」モードで一度にアーカイブした後で、「アーカイブなし磁気テープ」モードで、磁気テープ用の個々のレポートを作成できます。

## PAR 処理の概要

PAR 処理の仕組みは次のとおりです。

1. 関連するアサイメント処理を含めて、給与計算処理が作成されます。これらの処理では、PAR コードは、給与計算レポートに必要なデータを表す、ライブ・データベース・アイテム（ライブ表を指し示すアイテム）を評価します。PAR コードは、必要に応じてデータベース・アイテムのコンテキストを使用します。
2. PAR を「アーカイブ」モードまたは「アーカイブ付き磁気テープ」モードで実行した場合、データベース評価の結果は、FastFormula アーカイブ表（FF_ARCHIVE_ITEMS および FF_ARCHIVE_ITEM_CONTEXTS）に格納されます。
3. 「アーカイブ付き磁気テープ」モードまたは「アーカイブなし磁気テープ」モードで実行した場合には、PAR コードはアーカイブ・データベース・アイテムを評価することによりアーカイブ表から値を取り出し、磁気テープで提供されるレポートに値を入れます。

# 設定ステップの概要

## PAR 処理の設定方法

1. レポートおよびアーカイブ対象の従業員データを決定し、レポートのフォーマットを決定します。
2. 必要に応じてコンテキストを設定し、レポートで必要なデータを作成するために必要なアーカイブ・データベース・アイテムおよびライブ・データベース・アイテムを作成します。

   **関連項目：** アーカイブ用のデータベース・アイテムの作成 6-4 ページ

3. 「アーカイブ」モードまたは「アーカイブ付き磁気テープ」モードの場合には、アーカイブするデータベース・アイテムを決定するための Formula を記述します。「アーカイブ付き磁気テープ」モードおよび「アーカイブなし磁気テープ」モードの場合には、テープ・フォーマットで必要とされるフォーマットに文字列を設定し、ユーザーにエラー・メッセージおよび警告メッセージを提供する Formula を記述します。

   **関連項目：** Formula の記述 : 6-7 ページ

4. PAR がレポート作成のために処理するアサイメントおよびアサイメント処理を決定するパッケージ・プロシージャを記述します。

   **関連項目：** アサイメントおよびアサイメント処理のためのパッケージ・プロシージャの記述 : 6-7 ページ

5. ユーザーが PAR 処理を開始するための SRS（標準レポート発行）定義を提供します。

   **関連項目：** PAR 処理の SRS 定義の提供 : 6-9 ページ

6. PAY_REPORT_FORMAT_MAPPINGS_F 表の中に適切なエントリを入力して、カスタムのレポート、Formula およびパッケージ・プロシージャをシステムが識別できるようにします。

   **関連項目：** PAY_REPORT_FORMAT_MAPPINGS_F 表の行の挿入 : 6-10 ページ

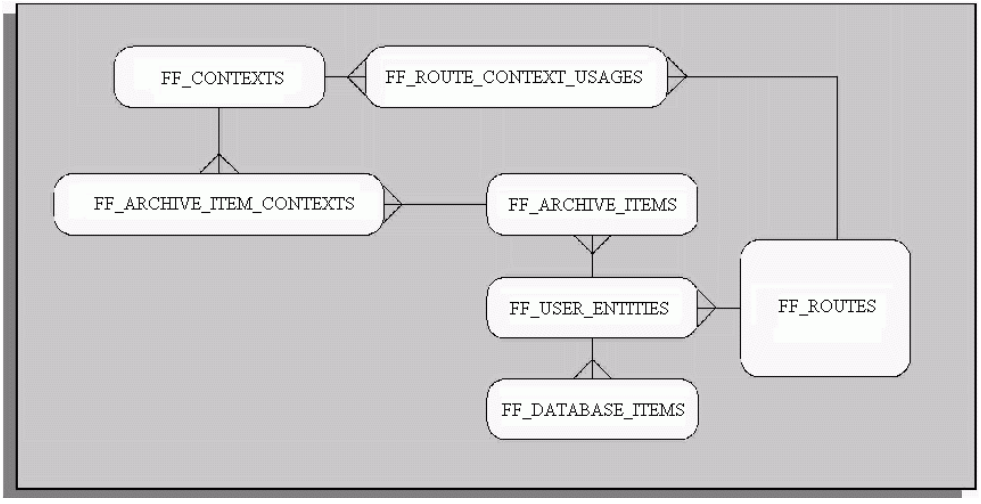
# アーカイブ用のデータベース・アイテムの作成

PAR はアーカイブ関数用に、ライブ・データベース・アイテム（ライブ表を指し示す）とアーカイブ・データベース・アイテム（アーカイブ・データを取り出すアーカイブ表を指し示す）の両方を使用します。アーカイブ・データベース・アイテムはそれぞれ、ライブ・データベース・アイテムと対応している必要があります。必要なライブ・データベース・アイテムおよびアーカイブ・データベース・アイテムがまだ存在していない場合には、ユーザーが作成する必要があります。

たとえば、Formula の中で A_INCOME_TAX_YTD を参照する場合、実際のデータベース・アイテム INCOME_TAX_YTD が存在している必要があります。PAR はこのライブ・データベース・アイテムを実行し、値をアーカイブ表 FF_ARCHIVE_ITEMS に入れます。

## アーカイブ・データベース・アイテムの作成 : 背景

次のエンティティ相関図は、PAR 表と、汎用人事管理システムのその他の表との関係を示しています。




FF_ARCHIVE_ITEMS 表は、PAR の実行時にどのデータベース・アイテムが評価されるかのスナップショットを記録します。

アーカイブ・データベース・アイテムの作成時には、アーカイブ・ルートも作成されます。これらのルートは、FF_ARCHIVE_ITEM_CONTEXTS および FF_ARCHIVE_ITEMS という 2 つの表からの簡単な SELECT 文である定義文を使用して FF_ROUTES の中に定義します。しかし、これらのルートは、ルートに渡されるコンテキストの数や、コンテキストのデータ型に基づいて定義する必要があります。しかし、ユーザー自身で定義するかわりに、そのまま利用できる、シード済のアーカイブ・ルートもあります。これについては次の項で詳しく説明します。

ルート・コンテキストの用途は FF_ROUTE_CONTEXT_USAGES 表の中に定義します。この定義を行う場合は、ライブ・ルートおよびアーカイブ・ルートが必要とするコンテキスト ID を FF_CONTEXTS から取り出し、新しいアーカイブ・ルートに基づいて新しいルート・コンテキスト用途を定義する方法をお薦めします。ルート・パラメータは必ず、新しいアーカイブ・ルートと、ユーザー・エンティティ ID のパラメータ名に基づいて定義されます。

より複雑なアーカイブ・ルートの例を紹介します。

```
l_text := 'ff_archive_items target,
   ff_archive_item_contexts fac,
   ff_archive_item_contexts fac1
where target.user_entity_id = &U1
and target.context1 = &B1 /* context assignment action id */
and fac.archive_item_id = target.archive_item_id
and fac.context = to_char(&B2) /* 2nd context of source_id */
and fac1.archive_item_id = target.archive_item_id
```

この比較的複雑なルートでも、基本となる単純な構造を見てとることができます。追加される各コンテキストは、FF_ARCHIVE_ITEM_CONTEXTS へのさらなる結合を表しているにすぎません。

## シードされた汎用アーカイブ・ルート

シードされた汎用アーカイブ・ルートには、2 つの種類があります。ただ 1 つのコンテキストを持つルート（ASSIGNMENT_ACTION_ID を使用）と、2 つのコンテキストを持つルートです。

### 1 つのコンテキストを持つルート

1 つのコンテキストを持つ汎用アーカイブ・ルートでは、そのコンテキストで 3 つのデータ型がサポートされるため、データベース・アイテムの自動ジェネレータを実行した場合は、このようなルートが 3 つ自動的に作成されます。

- 'T'（テキスト）データ型の FF_CONTEXT にマップされる文字コンテキストのルート。この名前は ARCHIVE_SINGLE_CHAR_ROUTE になります。
- 'N'（数値）データ型の FF_CONTEXT にマップされる数値コンテキストのルート。この名前は ARCHIVE_SINGLE_NUMBER_ROUTE になります。
- 'D'（日付）データ型の FF_CONTEXT にマップされる日付コンテキストのルート。この名前は ARCHIVE_SINGLE_DATE_ROUTE になります。

ARCHIVE_SINGLE_CHAR_ROUTE のテキストは次のようになります。

```
ff_archive_items target
    where target.user_entity_id = &U1
    and target.context1 = &B1
```

### 2 つのコンテキストを持つルート

2 つのコンテキストを持つ汎用アーカイブ・ルートの場合、データベース・アイテム自動ジェネレータは FF_ARCHIVE_ITEM_CONTEXTS 表を参照します。この表の CONTEXT 列は Varchar2(30) として格納されます。ここでは、FF_ARCHIVE_ITEMS に格納される最初のコンテキストは数値であり、アサイメント処理 ID であるものと想定されます。汎用アーカイブ・ルートの WHERE 句を次のようにデコードすることにより、このような「2 コンテキストのアーカイブ・ルート」を 1 つのみシードすることができます。

```
ff_archive_items target,
ff_archive_item_contexts context
ff_contexts ffc
where target.user_entity_id = &U1
and target.context1 = &B1
and target.archive_item_id = context.archive_item_id
and ffc.context_id = context.context_id
and context.context = decode(ffc.data_type,'T', &B2, 'D', fnd_date.date_to_
canonical(&B2),
to_char(&B2));
```

## アーカイブ・データベース・アイテム・ジェネレータの実行

アーカイブするデータベース・アイテムごとに 1 度ずつ、アーカイブ・データベース・アイテム・ジェネレータへのインタフェースを実行するプロシージャを何度かコールします。このプロシージャは次のとおりです。

```
procedure pay_archive_utils.create_archive_dbi(
     p_live_dbi_nameIN VARCHAR2(30),
     p_archive_route_nameIN VARCHAR2(30) DEFAULT NULL,
     p_secondary_context_nameIN VARCHAR2(30));
```

## データベース・アイテムのコンテキスト

標準の set_context プロシージャを使用して、コンテキストを必要とするデータベース・アイテムのためにグローバル・コンテキスト、またはアサイメント・レベルのコンテキストを設定します。INITIALIZATION_CODE は、たとえば PAYROLL_ID など、Formula のグローバル・コンテキストを設定します。ARCHIVE_CODE は、ASSIGNMENT_ID などのアサイメント・レベルのコンテキストを設定します。

**関連項目：** 例 : INITIALIZATION_CODE および ARCHIVE_CODE: 6-12 ページ

## Formula の記述

「アーカイブ」モードまたは「アーカイブ付き磁気テープ」モードで PAR を実行するためには、アーカイブ処理で使用するデータベース・アイテムを識別する Formula を記述します。「アーカイブ付き磁気テープ」モードまたは「アーカイブなし磁気テープ」モードで PAR を実行するためには、必要なフォーマットに文字列を設定し、警告およびエラーを提供するための Formula を記述する必要があります。

PAR 処理は、PAY_REPORT_FORMAT_MAPPING_F 表の REPORT_FORMAT 列の中にレポート用に存在しているエントリを使用して、適切な磁気テープ・フォーマットに関連する Formula を PAY_MAGNETIC_BLOCKS 表で検索します。

**関連項目：** PAY_REPORT_FORMAT_MAPPINGS_F 表の行の挿入 : 6-10 ページ

## アサイメントおよびアサイメント処理のためのパッケージ・プロシージャの記述

次のように 2 つのパッケージ・プロシージャを記述する必要があります。

- アーカイブで処理するアサイメント範囲を指定する RANGE_CODE プロシージャ。
- 処理するアサイメント処理を作成する ASSIGNMENT_ACTION_CODE プロシージャ。

### RANGE_CODE の例

このパッケージ・プロシージャは SELECT 文を戻します。この SELECT 文は、PAR でアサイメント処理を作成する必要のあるアサイメントを持つ person_id を戻します。

```
--
procedure range_cursor (pactid in number,
sqlstr out varchar2) is
begin
--
sqlstr := 'select distinct person_id
          from per_people_f ppf,
          pay_payroll_actions ppa
          where ppa.payroll_action_id = :payroll_action_id
          and ppa.business_group_id = ppf.business_group_id
          order by ppf.person_id';
--
end range_cursor;
```

**注意：** この文字列では :payroll_action_id というエントリをただ 1 度のみ使用する必要があります。また、文は order by person_id とする必要があります。

### ASSIGNMENT_ACTION_CODE の例

このパッケージ・プロシージャは、アサイメント処理をさらに限定して作成します。

```
--
procedure action_creation(pactid in number,
                          stperson in number,
                          endperson in number,
                          chunk in number) is
--
CURSOR c_state IS
   SELECT ASG.assignment_id assignment_id
     FROM per_assignments_f ASG,
          pay_payroll_actions PPA
    WHERE PPA.payroll_action_id = pactid
      AND ASG.business_group_id = PPA.business_group_id
      AND ASG.person_id between stperson and endperson
      AND PPA.effective_date between ASG.effective_start_date
                                 and ASG.effective_end_date
 ORDER BY ASG.assignment_id;
--
lockingactid number;
begin
  for asgrec in c_state loop
     --
     -- Create the assignment action to represent the person / tax unit
     -- combination.
     --
     select pay_assignment_actions_s.nextval
       into lockingactid
       from dual;
      --
      -- insert into pay_assignment_actions.
      hr_nonrun_asact.insact(lockingactid,asgrec.assignment_id, pactid,chunk, NULL);
   end loop;
end action_creation;
--
```

**注意：** このプロシージャには 4 つの値が渡されます。マルチスレッドで使用されるものであるため、ここでは作成を制限するために、開始個人および終了個人を使用する必要があります。同様に、チャンクも使用し、プロシージャに渡す必要があります。これにより実際に処理が作成されます。

## PAR 処理の SRS 定義の提供

PAR 処理は、「要求の発行」ウィンドウからユーザーが開始するバッチ処理です。ユーザーの処理用に SRS 定義を設定する必要があります。この定義のパラメータは次のとおりです。

**表 6-1 PAR 処理のパラメータ**

| パラメータ名： | 必須か？ |
|---|---|
| report_type | Yes |
| report_qualifier | Yes |
| start_date | No* |
| effective_date | No* |
| report_category | Yes |
| business_group_id | Yes |
| magnetic_file_name | No |
| report_file_name | No |
| legislative_parameters | No* |

* PAR 処理には start_date および effective_date が必要です。しかし、これらは標準パラメータへ入力するか、または特別な国別仕様パラメータ、START_DATE および END_DATE を使用することにより設定できます。これらの特別なパラメータは、START_DATE=<date> および END_DATE=<date> という形式で、legislative_parameters パラメータに渡されます。

## PAY_REPORT_FORMAT_MAPPINGS_F 表の行の挿入

PAY_REPORT_FORMAT_MAPPINGS_F 表にエントリを作成することにより PAR 処理を制御します。この表の各列は次のようになっています。

**表 6-2 PAY_REPORT_FORMAT_MAPPINGS_F 表の列**

| 列名 | タイプ | コメント |
|---|---|---|
| REPORT_TYPE | NOT NULL VARCHAR2(30) | レポートの短縮名です。例 :SQWL (State Quarterly Wage Listing の意味) |
| REPORT_QUALIFIER | NOT NULL VARCHAR2(30) | レポートの修飾名です。例 :SQWL の場合には州の名前を使用できます（テキサスやカリフォルニアなど）。 |
| REPORT_FORMAT | NOT NULL VARCHAR2(30) | PAY_MAGNETIC_BLOCKS 表への外部キーです。すべてのモードの実行で必要になります。 |
| 有効開始日 (EFFECTIVE_START_DATE) | NOT NULL DATE | |
| 有効終了日 (EFFECTIVE_END_DATE) | NOT NULL DATE | |
| RANGE_CODE | VARCHAR2(60) | アーカイブ中で処理するアサイメントの範囲を指定するためにユーザーが作成したパッケージ・プロシージャの名前です。コード例 : アサイメントおよびアサイメント処理のためのパッケージ・プロシージャの記述 : 6-7 ページを参照 |
| ASSIGNMENT_ACTION_CODE | VARCHAR2(60) | 処理するアサイメント処理を作成するためにユーザーが作成したパッケージ・プロシージャの名前です。コード例 : アサイメントおよびアサイメント処理のためのパッケージ・プロシージャの記述 : 6-7 ページを参照 |
| INITIALIZATION_CODE | VARCHAR2(60) | アーカイブの全期間を通じて必要となるグローバル・コンテキストを設定するパッケージ・プロシージャです。頻繁に使用されるものではありませんが、プロシージャを作成しておく必要があります (6-6 ページの「データベース・アイテムのコンテキスト」および 6-12 ページの「例 : INITIALIZATION_CODE および ARCHIVE_CODE」を参照)。この列に値が入力されていないと、PAR はアーカイブを実行しません。 |

**表 6-2 PAY_REPORT_FORMAT_MAPPINGS_F 表の列（続き）**

| 列名 | タイプ | コメント |
|---|---|---|
| ARCHIVE_CODE | VARCHAR2(60) | アーカイブ中に使用されるアサイメント処理レベルにコンテキストを設定します。INITIALIZATION_CODE のかわりに使用されることがあります。参照：データベース・アイテムのコンテキスト：6-6 ページおよび例：INITIALIZATION_CODE および ARCHIVE_CODE: 6-12 ページ |
| MAGNETIC_CODE | VARCHAR2(60) | 標準の汎用磁気テープ駆動 PL/SQL プロシージャです（5-21 ページの「磁気テープ処理」を参照）。PAR は磁気テープを作成するために、PAY_MAGNETIC_BLOCKS 表への外部キーとして REPORT_FORMAT を使用します。MAGNETIC_CODE に値が入力されていない場合、PAR は磁気テープを作成しません。 |
| REPORT_CATEGORY | NOT NULL VARCHAR2(30) | メディア・タイプのインディケータです。命名標準：<br>RT - リールからリール・テープ<br>SD - フロッピィ・ディスク<br>REPORT - 紙のレポート<br>ARCHIVE - アーカイブ |
| REPORT_NAME | VARCHAR2(60) | 「アーカイブ付き磁気テープ」、「アーカイブ」、および「アーカイブなし磁気テープ」モードの実行の場合は NULL のままになります。その他のモードで将来使用することが想定されています。 |
| SORT_CODE | VARCHAR2(60) | Oracle Report Writer を使用して提供されるレポートを処理する場合にのみ入力されます。処理する必要のある順序でアサイメント処理を戻す、ユーザーが作成したパッケージ・プロシージャの名前を入力します。 |

この表へのキーは、REPORT_TYPE、REPORT_QUALIFIER、REPORT_CATEGORY、EFFECTIVE_START_DATE および EFFECTIVE_END_DATE です。

# 例： INITIALIZATION_CODE および ARCHIVE_CODE

## INITIALIZATION_CODE

```
/*  名前： archinit
    目的： US 固有の初期化セクションを実行します。
 */
 procedure archinit(p_payroll_action_id in number) is
  jurisdiction_code   pay_state_rules.jurisdiction_code%TYPE;
  l_state             VARCHAR2(30);
begin
null;
end archinit;
```

## ARCHIVE_CODE

> **注意：** このコードは、アサイメント処理に従ってコンテキストを設定します。コンテキストを設定するには、set_context 関数を使用する方法と、PL/SQL コンテキスト表を使用する方法という 2 つの方法があります。コンテキスト表は、SOURCE_ID および SOURCE_TEXT の例のように、コンテキストが複数の値を持つ可能性がある場合にのみ使用されます。

```
/* 名前： archive_data
   目的： ZA 固有の従業員コンテキスト設定を実行します。
   */
procedure archive_data(p_assactid in number, p_effective_date in date) is
    asgid          pay_assignment_actions.assignment_id%type;
    l_count        number;
    l_context_no   number;
    aaseq          number;
    aaid           number;
    paid           number;
  cursor cursars is
    select distinct code
      from pay_za_irp5_bal_codes
     where code in (4001, 4002, 4003, 4004, 4005, 4006, 4007);
  cursor curclr is
    select distinct nvl(pet.element_information1, '&&&')
                                        element_information1
      from pay_element_types_f pet,
           pay_element_classifications pec,
           pay_assignment_actions paa,
           pay_payroll_actions    ppa
     where paa.assignment_action_id = p_assactid
```

```
      and pec.classification_name = 'Deductions'
      and pec.classification_id = pet.classification_id
      and ppa.payroll_action_id = paa.payroll_action_id
      and exists (select ''
                  from pay_assignment_actions paa2,
                       pay_payroll_actions    ppa2,
                       pay_run_results        prr
                  where paa2.assignment_id = paa.assignment_id
                    and paa2.payroll_action_id =
                                        ppa2.payroll_action_id
                    and paa2.assignment_action_id =
                                        prr.assignment_action_id
                    and prr.element_type_id = pet.element_type_id
                    and ppa2.effective_date between ppa.start_date
                                        and ppa.effective_date
                 );
  begin
      SELECT aa.assignment_id
             into asgid
        FROM pay_assignment_actions aa
       WHERE aa.assignment_action_id = p_assactid;

      l_context_no :=  pay_archive.g_context_values.sz;

      for i in 1..l_context_no loop

          pay_archive.g_context_values.name(i) := NULL;
          pay_archive.g_context_values.value(i) := NULL;
      end loop;
      pay_archive.g_context_values.sz := 0;
      l_count := 0;

/* Set up the assignment id, date earned and tax unit id contexts */

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
                                          'ASSIGNMENT_ID';
      pay_archive.g_context_values.value(l_count) := asgid;

      SELECT MAX(paa.action_sequence)
        INTO aaseq
        FROM pay_assignment_actions paa,
             pay_payroll_actions ppa,
             pay_action_classifications pac,
             pay_payroll_actions        ppa_arch,
             pay_assignment_actions     paa_arch
        WHERE
```

```
    paa_arch.assignment_action_id = p_assactid
    and paa_arch.payroll_action_id =
                                ppa_arch.payroll_action_id
    and paa.assignment_id = paa_arch.assignment_id
    AND paa.payroll_action_id = ppa.payroll_action_id
    AND ppa.action_type = pac.action_type
    AND pac.classification_name = 'SEQUENCED'
    AND ppa.effective_date between ppa_arch.start_date
                               and ppa_arch.effective_date
    and exists (select ''
                 from pay_payroll_actions ppa2,
                      pay_assignment_actions paa2,
                      pay_run_results prr,
                      pay_element_types_f pet
                where ppa2.time_period_id =
                                ppa.time_period_id
                  and ppa2.payroll_action_id =
                                paa2.payroll_action_id
                  and paa2.assignment_action_id =
                                prr.assignment_action_id
                  and prr.element_type_id =
                                pet.element_type_id
                  and ppa2.effective_date between
                             pet.effective_start_date and


                                pet.effective_end_date
                  and paa2.assignment_id = paa.assignment_id
                  and pet.element_name =
                                'ZA_Tax_On_Lump_Sums')
    and not exists (select ''
                      from pay_assignment_actions paa3,
                           ff_archive_items fai,
                           ff_user_entities fue
                     where paa3.assignment_id =
                                paa_arch.assignment_id
                       and paa_arch.payroll_action_id =
                                paa3.payroll_action_id
                       and paa3.assignment_action_id =
                                fai.context1
                       and fai.user_entity_id =
                                fue.user_entity_id
                       and fue.user_entity_name =
                                'A_PAY_PROC_PERIOD_ID'
                       and fai.value = ppa.time_period_id);
if aaseq is null then
  SELECT MAX(paa.action_sequence)
```

```
          INTO aaseq
          FROM pay_assignment_actions paa,
               pay_payroll_actions ppa,
               pay_action_classifications pac
          WHERE
            paa.assignment_id = asgid
            AND paa.payroll_action_id = ppa.payroll_action_id
            AND ppa.action_type = pac.action_type
            AND pac.classification_name = 'SEQUENCED'
            AND ppa.effective_date <= p_effective_date;
      end if;
      SELECT assignment_action_id, payroll_action_id
        INTO aaid, paid
        FROM pay_assignment_actions
       WHERE
         assignment_id = asgid
         AND action_sequence = aaseq;

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
                                          'ASSIGNMENT_ACTION_ID';
      pay_archive.g_context_values.value(l_count) :=aaid ;
      pay_archive.balance_aa := aaid;

      l_count := l_count + 1;
      pay_archive.g_context_values.name(l_count) :=
                                          'PAYROLL_ACTION_ID';
      pay_archive.g_context_values.value(l_count) :=paid ;

      for clrrev in curclr loop
        l_count := l_count + 1;
        pay_archive.g_context_values.name(l_count) :=
                                           'SOURCE_TEXT';
        pay_archive.g_context_values.value(l_count) :=
                                clrrev.element_information1;
      end loop;
      for sarrec in cursars loop
        l_count := l_count + 1;
        pay_archive.g_context_values.name(l_count) := 'SOURCE_ID';
        pay_archive.g_context_values.value(l_count) := sarrec.code;
      end loop;
--
      pay_archive.g_context_values.sz := l_count;
--
  end archive_data;
```

# 7

# Oracle Payroll のバランス

## Oracle Payroll のバランス

この章では、Oracle Payroll におけるバランスおよびバランス累計期間の定義ならびに使用方法について説明します。さらに、初期化バランスのロード方法についても説明します。ここでは、システムへのバランス累計期間の追加方法については説明しません。

### 条件

この章は、オラクル人事管理システムのデータベース設計に関して熟知されている方を対象としています。

バランスの設定および使用方法、あるいは従業員アサイメント、アサイメント処理、データベース・アイテム、または Oracle FastFormula における給与計算処理の概念に関する知識がない場合には、詳細は、『オラクル人事管理システム・ユーザーズ・ガイド』を参照してください。

給与計算処理のバランスについての詳細は、次のセクションも参照してください。給与計算処理 - バランスの作成と保守 : 5-9 ページ

## バランスの概要

Oracle Payroll では、バランスは給与計算の結果の累積であると定義付けられています。バランスには、項目、挿入、累計期間があります。

たとえば、バランスの「総支給額」は、全「支給項目」の処理結果の累計です。しかし、累計期間という考え方は Oracle Payroll に特有のものです。累計期間を使用することにより、様々な基準の組合せを使用してバランスの値を表示できます。このため、必要に応じて、ある従業員の現在の給与期間、または年累計の「総支給額」の値を表示することも可能です。Formula やレポートに使用する実績バランスおよび累計期間は、GROSS_PAY_ASG_PTD または GROSS_PAY_ASG_YTD になります。

一般的に、Oracle Payroll におけるバランスは、バランス値を入手するための計算ルールであると考えることができます。ほとんどの値は、データベースの中に明示的に保持されるわけではありません。このアプローチには次のような数多くの利点があります。(1) 新規バラ

ンスを定義して、いつでも好きなときに任意の挿入や累計期間に使用できます。(2) バランス値はデータベースに明示的に保存する必要がないので、貴重な記憶領域を占有し、データのアーカイブやパージに関する問題を引き起こすことがありません。

### バランス・タイプ

「総支給額」や「差引支給額」といったバランス名があります。バランス・タイプは常に数値「単位」を取りますが、場合によっては通貨コードを取ることもあります。

### バランス挿入

バランス挿入は、バランスに寄与する入力パラメータを定義します。たとえば、全支給項目タイプの結果値は、「総支給額」バランスに寄与します。バランス挿入はバランスに加算 (+) することも、バランスから減算 (-) することも可能です。

### バランス累計期間

バランス累計期間は、バランスのデータベース・アイテムの接尾辞によって識別されます。たとえば、'_YTD'_ は、そのバランス値が年累計であることを示します。バランス累計期間は、Oracle Payroll で事前定義されています。

### 事前定義バランス

事前定義バランスは、バランス・タイプとバランス累計期間の組合せを識別するのに使用される名前です。たとえば、GROSS_PAY_ASG_YTD があります。ユーザーが「バランス」ウィンドウを使用して新規バランスを定義すると、Oracle Payroll はユーザー選択のすべてのバランス累計期間に対して、データベース・アイテムを自動的に生成します。このため、ユーザーは任意の Formula 内の値に直接アクセスできます。バランスに関する詳細計算や詳細レポートでは、常に「事前定義バランス」を参照して値を返します。

## 最新バランス

給与計算処理のパフォーマンスを最適化するために、いくつかのバランス値がデータベース内に明示的に保持されています。これらの値は、「**最新バランス値**」と呼ばれます。給与計算処理を実行すると、最新バランス値がアクセスされ、更新されます。たとえば、ロールバックを行う場合などには、値は消去された後で再設定されます。これはまったくユーザーには表示されず、給与計算処理によって管理されます。

---

**注意：** レポート内でバランスの値を戻す必要があるときには、pay_balance_pkg.get_value というバランス関数を使用する必要があります。
参照 : レポートへのバランス値の組込み : 7-23 ページ

---

## 有効期限

最新バランスの重要な概念は、「有効期限」という概念です。たとえば、GROSS_PAY_YTD バランスについて考えてみましょう。次の会計年度に入ったら、値は 0（ゼロ）に戻るとお考えのことでしょう。バランスの「有効期限」は Oracle Payroll によって内部保持されています。また、次の会計年度に入ると有効になるコードがあります。

> **注意：** 給与計算に関して理論上は失効した場合でも、同じ給与計算で続けて更新されないかぎり、事前定義バランスが実際にデータベース上で 0（ゼロ）に設定されることはありません。このため、給与計算の後にも、失効になったと思っていたバランスがしっかりと表示される場合があります。ただし、値は前の値のままです。

## バランス・コンテキスト

ASSIGNMENT_ACTION_ID と BALANCE_TYPE_ID の組合せではレポートする必要のある個々のバランス値を一意に識別できない状況で、バランスをレポートする必要性が発生することもあります。たとえば、US 仕様では特定の州のバランス累計期間を保持する必要があるのに対して、UK 仕様では個別の税務署のバランス累計期間を保持する必要があります。

こうした要件はいずれも、特別なバランス・コンテキストを定義することによって満たされます。つまり、国別仕様 'C' コードで、バランス累計期間の一部として表示されます。

追加バランス・コンテキストのユーザー定義はまだサポートされていません。これは、給与計算処理の全体的なパフォーマンスに重大な影響を及ぼす可能性があるためです。大量の実行結果を累計する場合には特に、これらのコンテキストの定義に無効なコードがあると、実行速度が非常に遅くなる可能性があります。

### コンテキスト・バランス ー UK の例

コンテキスト・バランスをレポートするためには、ELEMENT_PTD および ELEMENT_ITD 累計期間を使用して関連するバランスを定義する必要があります。値をさらに識別するために必要となる追加コンテキストは PAY_RUN_RESULTS.SOURCE_ID から入手されます。これは、PAY_RUN_RESULT_VALUES 表に続いて PAY_RUN_RESULTS に結合するバランス挿入を行うことにより入手されます。

この値を使用すれば、PAY_ASSIGNMENT_LATEST_BALANCES -> PAY_BALANCE_CONTEXT_VALUES 方式を使用して選択を行うことができます。また、最新バランスがない場合には、UK ではファンクション・コールによって行えるルート・コード・コールによって行うことができます。

```
hr_gbbal.calc_element_ptd_bal(ASSIGNMENT_ACTION_ID,
                              BALANCE_TYPE_ID,
                              SOURCE_ID);
```

（または同じパラメータの calc_element_itd_bal）

# バランス累計期間

この章では、バランス累計期間とはどのようなものであるか、どのような働きをするか、各部分は Formula や「給与計算」とどのように影響しあうかということについて説明します。

バランス累計期間は、特定のバランスの値の計算方法を定義します。バランス累計期間は、バランス計算と関連付けられた固有の属性を持つエンティティでもあります。

## データベース・アイテムの接尾辞

データベース・アイテムの接尾辞によって、名前付きバランスの特定の累計期間が識別されます。「事前定義バランス」項目は、バランスと接尾辞の組合せです。たとえば、'GROSS_SALARY_ASG_YTD' の接尾辞 '_ASG_YTD' からは、総支給額バランスの値が 1 つのアサイメントに関して、年累計で計算されていることが分かります。

## ルート

バランス累計期間ルートは、FF_ROUTES 表に対する外部キーです。ルートは、ユーザーがバランスにアクセスしたときに返される値を定義する SQL コードの断片です。他のデータベース・アイテムの場合と同様に、テキストは FF_DATABASE_ITEMS 表の DEFINITION_TEXT 列に保持されています。

文の SELECT 句は必ず次のようになっています。

```
select nvl(sum(fnd_number.canonical_to_number(TARGET.result_value) * FEED.scale), 0)
```

このように、バランスは、ある一定の期間にわたって（本例の場合は、現会計年度の開始から現在まで）バランス・タイプ（本例の場合は、「総支給額」）を設定する実行結果値の合計であると定義付けることができます。

SQL 文自体は多数のルールに準拠している必要があります。例を次に示します。

```
        pay_balance_feeds_f FEED
       ,pay_run_result_values    TARGET
       ,pay_run_results          RR
       ,pay_payroll_actions      PACT
       ,pay_assignment_actions   ASSACT
       ,pay_payroll_actions      BACT
       ,pay_assignment_actions   BAL_ASSACT
where  BAL_ASSACT.assignment_action_id = \&B1
and    BAL_ASSACT.payroll_action_id    = BACT.payroll_action_id
and    FEED.balance_type_id            = \&U1
and    FEED.input_value_id             = TARGET.input_value_id
and    TARGET.run_result_id            = RR.run_result_id
and    RR.assignment_action_id         = ASSACT.assign_action_id
and    ASSACT.payroll_action_id        = PACT.payroll_action_id
and    PACT.effective_date between
          FEED.effective_start_date and FEED.effective_end_date
and    RR.status in ('P','PA')
```

```
and    PACT.effective_date >=
          (select to_date('06-04-' || to_char( to_number(
                  to_char( BACT.effective_date,'YYYY'))
           +  decode(sign( BACT.effective_date - to_date('06-04-'
               ||
to_char(BACT.effective_date,'YYYY'),'DD-MM-YYYY')),-1,-1,0)),'DD-MM-YYYY')
           from dual)
and    ASSACT.action_sequence <= BAL_ASSACT.action_sequence
and    ASSACT.assignment_id = BAL_ASSACT.assignment_id');
```

本例は、4 月 6 日を会計年度の開始とする UK 仕様のアサイメント・レベルの年累計バランスのルートを示したものです。

### コメント

ルートは、次の部分から構成されます。

1. アサイメントの全可能処理を返す
2. 可能なバランス挿入を識別する

   バランス挿入チェック
3. バランスを合計する期間を制限する

   有効期限チェック

> **注意：** 有効期限チェックとバランス挿入チェックの部分は特別な意味を持ちます。これについては後で説明します。

特定の表の別名は特定の意味を持つので、必ず使用してください。

- BAL_ASSACT 表は、「ソース」アサイメント処理、すなわちこのアサイメントに対する現在の処理です。
- ASSACT 表は、「ターゲット」アサイメント処理、すなわちこのバランスに挿入される結果に対する処理です。
- PACT 表は、「ターゲット」給与計算処理、すなわち ASSACT アサイメント処理の日付を定義するのに使用される処理です。
- BACT 表にアクセスすると、そのアサイメントが登場するすべての給与計算処理がリストされます。
- このバランス・タイプの FEED 表にアクセスすると、このバランスに挿入される可能性のあるすべての TARGET 入力パラメータがリストされます。
- バランスに挿入される実行結果は処理が必要です（'P' または 'PA'）。

- 複雑に見える副問合せは、現在の会計年度の開始、つまりバランス合計を開始したときを戻します。つまり、現在の会計年度の開始と、現在の処理順序の間が、結果としてバランスに挿入されることになります。

## 累計期間タイプ

累計期間タイプは、給与計算によるバランスの処理方法を決定します。事前定義の累計期間に関しては、給与計算の最高のパフォーマンスを実現するために、累計期間タイプが最適化されています。

累計期間タイプは、次のいずれかの値を取ります。

- **N** - バランス挿入も保存もしない。この累計期間タイプは、いつでも最新バランスを作成するわけではありません。この累計期間を持つバランスでは、バランスが実行されるたびに必ず SQL が再実行されます。
- **F** - バランス挿入はされるが、保存はされない。この累計期間タイプは、給与計算中に「メモリー内に」バランスを作成します。実行コードによってこのバランスへの挿入は実行されますが、データベース上に最新バランスが保存されることはありません。
- **R** - 実行レベル・バランス。この累計期間タイプは特に、現在の実行の総計をするバランスに用いられますが、適切なルートと一緒に使用する必要があります。データベース上には最新バランス値は一切保存されません。
- **A** - アサイメント・レベルでのバランス挿入および保存。この累計期間タイプは、アサイメント・レベルの最新バランスを作成して、PAY_ASSIGNMENT_LATEST_BALANCES 表に保存します。
- **P** - 個人レベルでのバランス挿入および保存。この累計期間タイプは、アサイメント・レベルの最新バランスを作成して、PAY_PERSON_LATEST_BALANCES 表に保存します。

## バランス挿入チェック・タイプ

バランス挿入チェック・タイプは、給与計算中に用いられるバランス挿入チェック方法を制御します。このタイプは、バランスに実行結果を挿入するかどうかを決定することによって、メモリー内バランスを最新に保つために用いられます。このタイプは、次の値を取ることができます。

- **NULL** これはデフォルト値で、既存のバランス挿入によって含められたすべての実行結果値がそのバランスに挿入されることを意味します。
- **P** 「給与計算」は、その累計期間の expiry_checking_code 列に定義されているパッケージ・プロシージャを実行します。失効フラグ・パラメータは、バランス挿入が発生するか否かを示します。
- **E** 等価バランス挿入チェックが行われます。つまり、メモリー内バランス・コンテキスト値と UDCA（ユーザー定義コンテキスト領域）に保持されているコンテキストが一致する場合には、バランス挿入が発生します。

次の追加タイプは US 国別仕様のバランスでのみ使用されます。

- **J** 管轄区域チェックが行われます。
- **S** 項目バランス挿入チェックが行われます。
- **T** 'E' および 'S' バランス挿入チェック・タイプの組合せです。
- **M** 'S'、'J' および 'E' のバランス挿入チェック・タイプの組合せです。

## 失効チェック・タイプ

最新バランスは、累計期間によって決定されるタイミングで失効します（つまり 0（ゼロ）に戻ります）。たとえば、YTD（年累計）のバランスは、年末に失効します。

給与計算処理により、ロードされているすべてのバランスは、失効チェック・タイプに従ってチェックを受けます。

- N - 失効非使用。バランスが 0（ゼロ）に設定されることはありません。
- P - 給与処理レベル。これらのタイプに関しては、各所有処理 / バランス累計期間の失効チェック結果のリストが保持されます。

  失効チェック・コードは、一度こうした組合せに対してコールされたら、同じ組合せを持つ他のバランスに対して再度チェックする必要はありません。このため、データベースを何度もコールすることがなくなります。

  失効チェックはバランス・コンテキストから独立しています。このため、バランス・コンテキストのリストは、失効チェック・コードには渡されません。
- A - アサイメント処理レベル。仮定は一切行われず、失効チェック・コードが常にコールされます。失効チェックはバランス・コンテキストに依存しています。このため、バランス・コンテキストのリストは、失効チェック・コードに渡されます。
- D - 有効期限。有効期限チェック機能は、失効チェックされるバランスのバランス累計期間 / バランス・コンテキストの組合せを調べ、メモリー内リストをスキャンして、同じ組合せを持つバランスがすでに失効チェックされているかどうか確認します。

  すでに失効チェックされている場合には、メモリー内バランスに保存されている有効期限が取り出されます。

  失効チェックはバランス・コンテキストに依存しています。このため、バランス・コンテキストのリストは、失効チェック・コードに渡されます。

# Oracle Payroll の初期化バランスのロード

この章では、既存の給与システムからの初期化バランス値のロードを容易に行うために使用することのできる Oracle Payroll の機能について説明します。

## 概要

Oracle Payroll を初めて導入される場合でも、以前のリリースからアップグレードされる場合でも、自国の国別バランスの初期値を設定する必要があります。Oracle Payroll で正確な法定控除項目計算を実施するためには、これらのバランスの初期値が正しいことが必要不可欠となります。

この項では、給与計算処理を開始する前に行う、こうしたバランス初期値の設定およびロードの方法を説明します。給与計算処理の開始後は、将来追加ユーザー・バランスを定義する場合にも、必要に応じてこの処理を繰り返してください。

**警告：** この初期バランスをロードするための手順は、エンド・ユーザーがバランスを調整するために実行する手順とはまったく異なります。バランス調整を行うためにバランス・ロードの方法を使用しないでください。

### Oracle Payroll におけるバランスおよびバランス調整

Oracle Payroll では、バランスは給与計算の結果の累積のことです。バランスには、項目、挿入、累計期間があります。特定のバランスに挿入される結果は、「バランス挿入」として知られており、合計に加算 (+) することも、合計から減算 (-) することも可能です。バランスのロード処理によって、正しい実行結果が計算および挿入され、アップロード日から有効になる初期値が設定されます。

バランスは、バランス挿入を目的とした実行結果から直接計算されます。このアプローチでは、実行結果とバランス値がいつも同期がとられるため、追加情報をデータベースに保存および保持する必要はありません。事実上、バランスの定義はバランス値を返すために実行される「計算」の定義になります。

事前定義バランスに挿入される実行結果は、通常は、給与計算中のエレメントの処理結果になります。ただし、バランス値を手動で調整する場合もあります。これを行うには、エレメントに「バランス調整」のエントリを作成します。バランス調整をオンラインで行うときには、エレメントに対して単一の処理済実行結果を作成するのが有効です。この実行結果は、エレメントが通常挿入されているすべてのバランスに自動的に挿入されたり、それらのバランスを自動的に調整します。このように、ユーザーは、影響を受けたすべてのバランスに合わせた調整をカスケード化することが可能です。

**注意：** オンラインでバランス調整を行う場合は、適切なエレメントと入力パラメータを選択するように注意してください。間違えて入力してしまった場合には、必ず戻って入力を削除し、調整を再入力してください。バランス調整は、「給与」または「アサイメント処理」ウィンドウから削除します。

## ステップ

バランス初期値のロードは、基本的に次の 3 つのステップで行われます。

1. 各特定のバランスに挿入するエレメントと入力パラメータの定義
2. 次の表のバランス初期値の設定

   PAY_BALANCE_BATCH_HEADERS
   PAY_BALANCE_BATCH_LINES
3. 「初期化バランスのアップロード」処理の実行
   - SRS ウィンドウを使用。
   - 必要に応じて、検証、転送、取消、パージの各モードを使用。

## バランス・ロード処理

初期化バランスのロード・プロセスを実行する場合は、特定の日付（**アップロード日**）に関連する各バランスの値を設定します。この処理ではデートトラックされるバランス・エントリまたは「調整」が作成されるため、アップロード日以降の国別バランスは正確なものになります。この日以降のバランス情報の保守は、システム管理となるか、バランス調整を使用して管理されます。

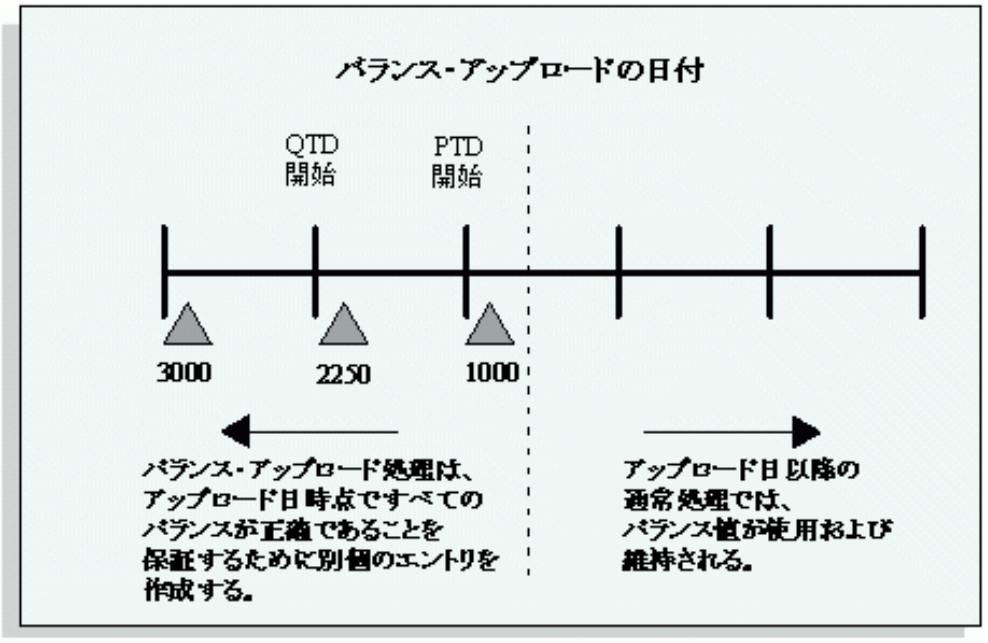
次の例について考えてみましょう。本例は、ある従業員の総支給額バランス値の 3 種類の累計期間を示したものです。

- 総支給額期間 10000.00
- 総支給額四半期 3250.00
- 総支給額年 6250.00

バランスのロード処理では、各エントリに必要とされる実績値と、これらのエントリの有効日が計算されます。計算の結果、次の 3 種類のバランス・エントリが作成されます。

- _PTD バランス・エントリ値 = 1000.00
- _QTD バランス・エントリ値 = 2250.00
- _YTD バランス・エントリ値 = 3000.00

**図 7-1 バランスのロード**




その結果として、個々のエントリの累積値は各バランスの初期要件に適ったものになります。

- 総支給額期間＝ 1000.00
- 総支給額四半期＝ 1000.00 ＋ 2250.00 ＝ 3250.00
- 総支給額年＝ 1000.00 ＋ 2250.00 ＋ 3000.00 ＝ 6250.00

## 最新バランス

給与計算のパフォーマンスの向上を図るため、Oracle Payroll では「最新バランス値」が設定および保持されます。これらの値が設定されていない場合には、バランスの実行結果を合計することによってバランス値が作成されます。多数のアサイメントが値を持たない場合には、初回の給与計算に重大な影響が及ぶ恐れがあります。このため、初回の給与計算の前に最新バランスをロードすることが、パフォーマンスには非常に重要です。

**注意：** メモリー内で使用されるが保存はされないバランスなど、最新バランスを作成できないバランスもあります。

初期ロード処理に含めるバランスや累計期間を決定するときには、給与計算に使用されるバランスについて考えてみます。たとえば、給与計算で bal_YTD バランスを使用し、アップロード処理で bal_PTD のみをロードした場合には、bal_PTD の最新バランス値は存在しますが、bal_YTD の最新バランス値は存在しません。最初の給与計算で bal_YTD を評価する必要があります。

通常の給与計算では、最新バランス値は、事前定義バランスを使用する最新アサイメント処理と関連付けられています。バランスのアップロード処理では、アップロード日前に多数のバランス調整エントリを作成することによって、最新アサイメント処理のシミュレーションが試行されます。

**注意：** 事前定義バランスにコンテキストが含まれている場合には、作成される最新バランスと矛盾しないコンテキスト値を持つバランス調整給与支払計算においてのみ、最新バランスの作成が可能です。

Oracle Payroll では、各バランス調整エントリは、個別のアサイメント処理であると見なされます。これらの調整は、日付順に（最早日から）実行されます。アサイメント処理番号が最も大きい最新バランス調整を使用して、最新バランスが作成されます。

## 初期化バランスに挿入するエレメントの設定

特定のバランスに関してはバランス挿入を含む複雑な Web が存在する可能性があるため、特定のバランスの初期値を設定するための簡単な方法が提供されています。(1) ユーザーは特定のバランス 1 つ 1 つに挿入する特別なエレメント入力パラメータを必要とする、(2) ユーザーは各バランスを個別に設定する、ということが基本原理となっています。

### 国別バランスを初期化するエレメント

Oracle Payroll には、自国の全国別バランスの初期値を設定するのに必要な事前定義のエレメントと入力パラメータが搭載されています。

**注意：** US ユーザーは、特別な PL/SQL スクリプト（paybalup.pkb）を実行して、事前定義の国別バランスに挿入するエレメントおよび入力パラメータを作成する必要があります。このスクリプトは、SRS 処理（初期化バランス体系の作成）として登録されています。これらのエレメント 1 つ 1 つに対して、バッチ・ラインを作成してください。

その他の国のユーザーは、初期化する必要のある国別バランスに挿入する事前定義エレメントをリンクするだけですみます。

## ユーザー定義のバランスを初期化するエレメント

他のバランスについてはすべて、各初期化バランスの入力値を提供するエレメントを設定する必要があります。バランスの初期値挿入用のエレメントの設定に関しては、いくつかのルールがあります。

### エレメント

- 開始日が 01-JAN-0001 であること。

  このルールは、バランスに挿入されるエレメントと入力パラメータを常に使用可能にすることによって、検証を簡略化します。

- 「バランスの初期値設定」という分類であること。

  この分類は、バランスを定義するときに使用可能な分類のリストには含まれていません。このタイプのエレメントに関しては、手動バランス挿入しか設定できません。

- 「バランス調整用」であること。
- 変動処理型であること。
- 給与計算で処理できること。

### 入力パラメータ

- 開始日が 01-JAN-0001 であること。
- 各入力パラメータは 1 つのバランスのみに挿入されること。

  多数のバランスの初期値を設定する必要がある場合には、各入力パラメータが個別のバランスに挿入されている単一のエレメントに対して、複数の入力パラメータを定義できます。

### エレメント・リンク

- 開始日が 01-JAN-0001 であること。
- 基準は必ず「全給与にリンク - 'Yes'」であること。

### サポートされているバランス

初期化処理でサポートされているバランスはすべて、アサイメント・レベルで設定されます。個人レベルのバランスは、あらゆるアサイメントの値を累計することによって間接的に設定されます。

## バランス初期値の設定

バランスの初期値は、多種多様なソースからロードできます。たとえば、前のバージョンの Oracle Payroll や別の給与システムから移行することも、この情報を別のシステムに保持することも考えられます。

バランスの初期値のロード処理を標準化するため、オラクル人事管理システムには 2 種類のバッチ・インタフェース表が搭載されています。これらの表に直接情報をロードして、値を手動でレビュー、更新および挿入することも可能です。このため、値の設定は非常に柔軟に行うことができます。また、複数の個別バッチを論理グループとしてロードするように定義および管理することもできます。

**PAY_BALANCE_BATCH_HEADERS**

| 名前 | NULL? | タイプ |
|---|---|---|
| BUSINESS_GROUP_ID | | NUMBER(15) |
| PAYROLL_ID | | NUMBER(9) |
| BATCH_ID | NOT NULL | NUMBER(9) |
| BATCH_NAME | NOT NULL | VARCHAR2(30) |
| BATCH_STATUS | NOT NULL | VARCHAR2(30) |
| UPLOAD_DATE | NOT NULL | DATE |
| BATCH_REFERENCE | | VARCHAR2(30) |
| BATCH_SOURCE | | VARCHAR2(30) |
| BUSINESS_GROUP_NAME | | VARCHAR2(60) |
| PAYROLL_NAME | | VARCHAR2(80) |

各バッチでは、アップロード中の給与計算とアップロード日が示されます。各バッチを一意に識別するために、たとえば次の表で示すように、その他の識別子を設定できます。

| バッチ名 | バッチ参照 | バッチ・ソース | 給与計算 | アップロード日 |
|---|---|---|---|---|
| 週次給与計算 | 0001 | SQL*Loader | Pay1 | 01-Jan-1995 |
| 週次給与計算 | 0002 | SQL*Loader | Pay1 | 01-Jan-1995 |
| 月次給与計算 | 0003 | SQL*Loader | Pay2 | 01-Jan-1995 |
| 2 回 / 月次給与計算 | 0001 | スクリーン | Pay3 | 01-Aug-1995 |

**PAY_BALANCE_BATCH_LINES**

| 名前 | NULL? | タイプ |
|---|---|---|
| ASSIGNMENT_ID | | NUMBER(10) |
| BALANCE_DIMENSION_ID | | NUMBER(9) |
| BALANCE_TYPE_ID | | NUMBER(9) |
| PAYROLL_ACTION_ID | | NUMBER(9) |
| BATCH_ID | NOT NULL | NUMBER(9) |
| BATCH_LINE_ID | NOT NULL | NUMBER(9) |
| BATCH_LINE_STATUS | NOT NULL | VARCHAR2(30) |
| VALUE | NOT NULL | NUMBER |
| ASSIGNMENT_NUMBER | | VARCHAR2(30) |
| BALANCE_NAME | | VARCHAR2(80) |
| DIMENSION_NAME | | VARCHAR2(80) |
| GRE_NAME | | VARCHAR2(60) |
| JURISDICTION_CODE | | VARCHAR2(30) |
| ORIGINAL_ENTRY_ID | | NUMBER(15) |

各バッチには、アサイメント詳細、バランスおよび各累計期間の値が含まれたバッチ・ライン・セットが含まれています。特定のバランスの他のコンテキストを含めることも可能です。

| アサイメント | バランス | 累計期間 | 値 |
|---|---|---|---|
| 101 | 総支給額 | PTD | 1000.00 |
| 101 | 総支給額 | QTD | 3250.00 |
| 101 | 総支給額 | YTD | 6250.00 |
| 101-2 | 総支給額 | PTD | 750.00 |

**注意：** これらの表は、それぞれの項目について、システム ID（assignment_id など）またはユーザー ID（assignment_number など）のいずれかのサポートを示しています。このため、バッチ表の作成において最大限の柔軟性が提供されます。

両方が指定されている場合には、システム ID がユーザー ID を上書きするというルールがあります。これらの表の設定時に指定することができるシステム ID とユーザー ID の一覧を次に示します。

| システム ID | ユーザー ID |
|---|---|
| BUSINESS_GROUP_ID | BUSINESS_GROUP_NAME |
| PAYROLL_ID | PAYROLL_NAME |
| ASSIGNMENT_ID | ASSIGNMENT_NUMBER |
| BALANCE_DIMENSION_ID | DIMENSION_NAME |
| BALANCE_TYPE_ID | BALANCE_NAME |
| ORIGINAL_ENTRY_ID | |
| GRE_NAME（US のみ） | |
| JURISDICTION_CODE（US のみ） | |

バッチ処理中にエラーが発生した場合には、エラー・メッセージが、H（ヘッダー）または L（ライン）の source_type とともに PAY_MESSAGE_LINES 表に書き込まれます。

# 初期化バランスのアップロード処理

SRS ウィンドウから初期化バランスのアップロード処理を実行して、バッチ表から値をアップロードします。この処理は、次の 4 つのモードのうちのいずれかで実行できます。

- 検証
- 転送
- 転送取消
- パージ

## 前提条件

アップロード日には、バッチ内のすべてのアサイメントがバッチ・ヘッダーに指定されている給与計算に属している必要があります。

給与計算には、アップロード日より前にバランスの初期値設定を行えるだけの十分な期間が必要です。

初期化バランスのアップロード処理では、GRE または法人企業など、その他特定の基準は検証されません。この情報の検証はユーザーの責任です。

---

**注意：** 検証処理には、ユーザー定義の追加検証プロシージャをユーザー独自のバランスに適用できるようにする、事前定義フックが含まれています。こうしたプロシージャには、*validate_batch_line* という名前を付ける必要があります。

この処理では、有効なデータのチェックは行われますが、設定は行われません。

---

## モード

### 検証モード

この処理を実行する前にバッチ表が検証されることはありません。この処理では、PAY_BALANCE_BATCH_LINES 表のデータの妥当性が検査されますが、それらのデータはオラクル人事管理システムのデータベースには転送されません。有効ラインには V（検証完了）、エラーのあるラインには E（エラー）とそれぞれマークが付けられ、エラー・メッセージが PAY_MESSAGE_LINES 表に送られます。

検証処理は、次の 2 つの段階に分けられます。

- 最初の段階では、バッチ表内のデータの整合性がチェックされます。
- 2 番目の段階では、必要なすべてのバランス調整エントリの作成が可能かどうかがチェックされます。

検証処理ではまた、システム ID エントリが表内に挿入されます。このため、後続のすべての処理がシステム ID にアクセスできます。

バッチ・ラインはすべて個別に検証され、処理の終わりに個別のステータスを表すマークが付けられます。

### 転送モード

転送モードでは、バッチ表内のデータの整合性と全システム ID の存在を確保するために、妥当性チェックの最初の段階が繰り返されます。

この処理では、各アサイメントに必要とされるバランス調整エントリが計算されます。値が共有されている上に、実績エントリがそのアサイメントに対して作成されている場合には、このリストがチェックされ、集計されます。これは、バッチ内の各アサイメントに対して繰り返されます。完了した転送には、'T'（転送完了）というステータスを表すマークが付けられます。

---

**注意：** アサイメントのいずれかのラインにエラーがある場合には、そのアサイメントのラインは人事管理システム・データベースには 1 行も転送されません。失敗が処理中のバッチ・ラインに対するメッセージ表にログされ、バッチ・ラインには I（無効）とマークが付けられます。

---

調整の値が 0（ゼロ）の場合には、エントリは 1 つも作成されません。例 :

Balance_PTD ＝ 500

Balance_QTD ＝ 500

値がすでに PTD によって設定されているので、QTD 累計期間にあわせて調整する必要はありません。

ロードするデータが大量にありそうなので、完了した作業を保存し、必要とされるロールバック・セグメントの数を減らすために、作業が定期的にコミットされます。

---

**注意：** コミット・サイズは、PAY_ACTION_PARAMETERS_ の CHUNK_SIZE パラメータによって指定されます。CHUNK_SIZE のデフォルトは 20 個の完了アサイメントです。

これは、他の給与計算処理でコミット頻度を決定する場合に使用されるパラメータと同じです。

---

バッチ処理が部分的に完了した場合には、そのバッチを再発行できます。この場合、「転送」されなかったバッチ・ラインを持つアサイメントのみが再度処理されます。処理の途中で失敗してしまった場合、たとえば、表領域が不足した場合にも、バッチ処理を再起動することができます。

**転送取消**

このモードでは、転送処理によって作成されたバランス調整エントリがすべて削除され、バッチ・ラインのステータスが U に戻ります。

> **注意：** バッチ表内のデータは保持されます。誤った値が含まれているバッチ・ラインを訂正して、転送を繰り返すことができます。

**パージ**

現在のステータスに関係なく、バッチ内のすべてのデータをパージします。1 つのバッチをパージすると、メッセージ、バッチ・ライン、バッチ・ヘッダーがすべて削除されます。このため、ユーザーは、バッチの転送が完了したらすぐに、領域を再生することができます。

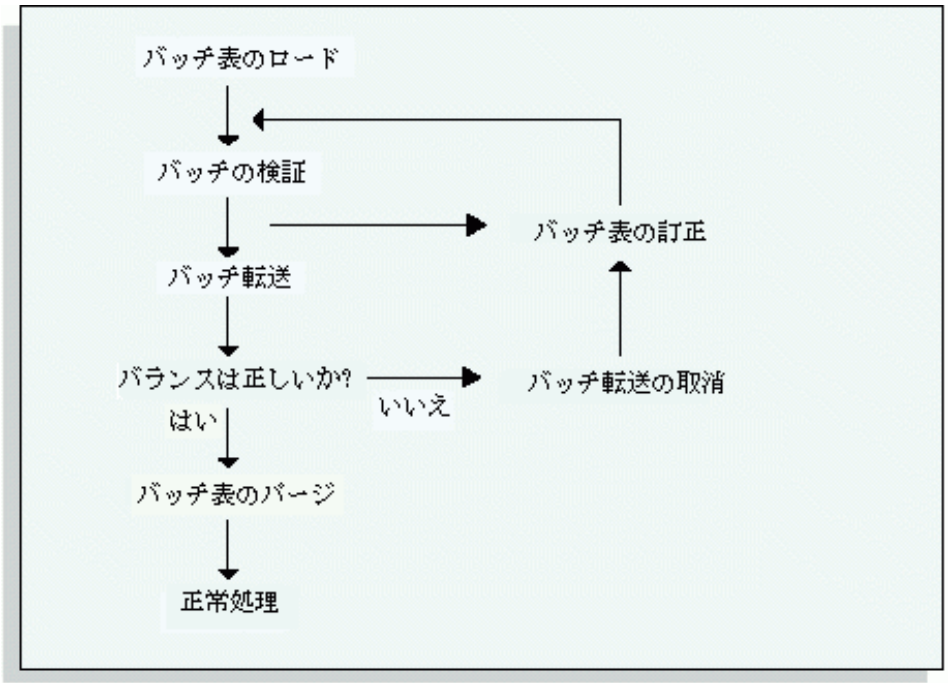
パージ・モードは、バッチ内のすべてのアサイメントのバランスが人事管理システム・データベースに入力されたことが確実にわかっている場合にのみ使用してください。

> **警告：** 一度バッチをパージすると、そのバッチのすべてのエントリが削除されます。この処理は取消しできません。

## プロセス・フロー

初期化バランスのロードにおいて、これらのモードが通常使用される順番を、次の図で示します。

**図 7-2 プロセス・フロー**




## エラー状況

発生したエラーはすべて、検証中のオブジェクト（バッチ自体または個々のバッチ・ライン）に対するメッセージ表に記録されます。そのバッチまたはバッチ・ラインに対して設定されるステータスは、他のバッチ・ラインのステータスはもちろん、処理の実行モードによって決まります。

### バッチ・ライン・ステータス

各バッチ・ラインのステータスは、次のいずれかを取ります。

- V ― 有効。バッチ・ラインは OK です。
- E ― 無効。バッチ・ラインにはエラーがあります。
- T ― 転送完了。バッチ・ラインの転送は完了しました。

### バッチ・ステータス

バッチのステータスは、バッチ内のバッチ・ラインのステータスによって決まります。

- T − 転送完了。バッチ内のすべてのバッチ・ラインの転送が完了しました。
- P − 部分転送。バッチ・ラインの一部のラインが転送されました。
- V − 有効。バッチ内のラインはすべて有効ですが、1 行も転送されませんでした。
- E − 無効。バッチ内のラインのいくつかが無効なので、1 行も転送されませんでした。

## 検証上の問題点

チェックする必要のある一般的な問題が 2 つあります。

バランス累計期間に対する調整要求が不正確な場合があります。たとえば、アサイメントに次のようなアップロード要求がある場合などです。

- <Balance>_QTD = 1500.00
- <Balance>_YTD = 1000.00

YTD の値は QTD の値よりも小さいものです。バランスがしだいに減少するのであれば、これは有効かもしれません。しかし、通常はバランスは増加するものです。このため、バランスが減少する場合はチェックすることをお薦めします。

第二に、調整処理で適切な調整実行日が見つからなかった場合には、無効調整エラーが発生する可能性があります。このエラーの原因は、処理中のバランス累計期間によって異なります。

しかし、どのような場合でも、すべてのビジネス・グループ詳細が正しいこと、バランスの設定が行えるだけの給与計算期間があることを確認するのが得策です。各アサイメント・バランスに使用されている日付をチェックするには、次の SQL を使用します。

```
select BL.dimension_name,
pay_balance_upload.dim_expiry_date
(BH.business_group_id
,BH.upload_date
,BL.dimension_name
,BL.assignment_id
,BL.gre_name
,BL.jurisdiction_code
,BL.original_entry_id)    expiry_date
from pay_balance_batch_headers BH
,pay_balance_batch_lines   BL
where BH.batch_name    = '&Batch_Name'
and BL.batch_id      = BH.batch_id
and BL.assignment_number = '&Assignment_Number'
and BL.balance_name  = '&Balance_Name'
;
```

有効期限が '31-DEC-4712' に設定されていると、調整日は見つかりません。

## バランスの初期化ステップ

データの設定方法に関する簡単なチェック・リストを次に示します。

1. その年の初めに遡る期間の給与計算を Oracle Payroll で作成します。全従業員をオラクル人事管理システムに入力して、これらの給与計算に対するアサイメントを割り当てます。

   **注意：** 次のステップは US ユーザーのみを対象としています。その他の国のユーザーは事前定義バランス・ロード・エレメントのリンクを定義するのみです。

2. 「要求の発行」ウィンドウから、「初期化バランス体系の作成」処理を実行し、バッチ名をパラメータとして選択します。この処理では、各バッチに対して次のものが作成されます。

   - 各バランスやなんらかのコンテキストの値を保持するための入力パラメータと、作成されたすべての入力パラメータを保持できる数の、「初期化バランス」という特別分類のあるエレメント。
   - これらのエレメントに必要なリンクとバランス挿入

3. ユーザー独自の支給項目と控除項目のバランスを初期化する必要のある、その他のエレメントを作成します。

   - 前述の要件に準拠する。参照 : 初期化バランスに挿入するエレメントの設定 : 7-11 ページ
   - 複数の入力パラメータを使用することによりエレメントの数を減らす。
   - 各入力パラメータに対して 1 つのバランス挿入を定義する。

   **注意：** 1 つのバランスに設定できるバランス初期値は 1 つのみです。

   同じ「アップロード日」を持つバランスに、1 つのエレメントに対する複数の入力パラメータを挿入する必要があります。

4. バランスの初期化を管理するために従業員をバッチとしてグループ化します。各バッチを識別するヘッダーを入力します（これらのヘッダーは PAY_BALANCE_BATCH_HEADERS 表に入れられます）。各ヘッダーには次の情報が含まれます。

   - ビジネス・グループ名と給与
   - バッチ名と ID 番号

- アップロード日 : 現在のシステムにおいてバランスが正しく、転送の準備が整っている日付

例 :

```
Batch Name   Business Group   Payroll Name   Upload Date
Upload 1     BG name          Full Time 1    13-AUG-1995
```

5. 転送する各バランスのバッチ・ラインを作成します（これらのラインは、PAY_BALANCE_BATCH_LINES 表に入れられます）。バッチ・ラインには、次の情報が含まれています。

   - 従業員アサイメント番号
   - バランス名と累計期間（四半期累計や年累計など）
   - バランス値
   - バランス・コンテキスト（該当する場合）。US ユーザーに関しては、コンテキストには GRE や管轄区域（連邦政府、州または地方）が含まれる場合があります。

   **注意 :** プロセスは、ユーザーが作成したバランス挿入定義を使用して、どのエレメント入力パラメータを使用するかを決定します。

   例 :

```
Asg. Number   Balance     Dimension         Value
60001         Salary      PTD               700
60001         Salary      QTD               1400
60001         Salary      YTD               2400
60001         Tax Paid    PTD               2200
60001         Tax Paid    QTD               2400
60001         Tax Paid    YTD               2400
```

   **注意 :** Tax Paid YTD の値は QTD の値と同じであるため、必要ありません。しかし、最新バランスの値を作成して、最初の給与計算処理のパフォーマンスを向上させるために、このバランスが含められています。

6. 「要求の発行」ウィンドウから、「初期化バランスのアップロード」処理を実行します。この処理をパラメータとして実行する際のモードを選択します。使用可能なモードは次のとおりです。

   - **検証**
     - バッチ・ラインの検証は行われますが、転送は行われません。
     - エラー・メッセージが PAY_MESSAGE_LINES に送られます。

- **転送**
  - バッチ・ラインが検証および転送されます。
  - アサイメントのいずれかのラインにエラーがある場合には、そのアサイメントのラインは 1 行も転送されません。
- **取消**

  データベースからバランス初期化エントリが削除され、バッチ・ライン表のそれらのラインに 'U' というマークが付けられます。
- **パージ**

  付けられているマークに関係なく、バッチ・ライン表内のすべてのラインがパージされます。

---

**注意：** パージ・モードは、バッチ内のすべてのアサイメントのバランスが人事管理システム・データベースに入力されたことが確実にわかっている場合にのみ使用してください。

---

# レポートへのバランス値の組込み

この項では、ユーザーが問合せおよびレポート作成ツールで使用するためにバランス値にアクセスできるようにする、バランス関数の PL/SQL インタフェースについて説明します。

**UK ユーザー** - 参照 : レポートへのバランス値の組込み（UK のみ）: 7-31 ページ

**提案 :** 同じバランス値を別のレポートの中で何度も報告する場合には、レポート表を作成することを検討してください。PL/SQL スクリプトにバランス関数を含めるだけで、この表に挿入することができます。

## 利点

PL/SQL 関数を使用したバランス値の取出しには、次のようないくつかの利点があります。

- この関数は、フォームまたは SRW2 レポートから簡単にコールできます。
- 最新バランス値（存在している場合）にアクセスできます。これによって、パフォーマンスが必然的に最適化されます。

# バランス関数

この関数へのインタフェースは、柔軟性が高く、操作も簡単です。関数内のコンテキストのハードコードされる知識は最小限に抑えられるので、バランス関数は次のようにして制御されます。

- 関数のコール前に、使用するコンテキストを設定するための別の PL/SQL 関数がコールされます。これらのコールは、パッケージ・レベルの PL/SQL 表に保持されます。このため、コンテキスト情報がハードコードで指定されていなくてもバランス関数は機能できます。さらに、複数のバランスをコールする場合でもクライアント / サーバー・コールの回数を減らすことができます。
- 'C' バランス・ユーザー・イグジットは、日付処理とアサイメント処理の 2 つのモードで機能します。バランス関数はモード・パラメータを渡しません。モードは PL/SQL オーバーロード機能を使用して解決されます。このため、インタフェースが簡略化されます。

PL/SQL コードは 1 つのパッケージに常駐します。

```
pay_balance_pkg
```

**- プロシージャ : コンテキストの初期化 :**

```
procedure set_context (p_context_name in varchar2, p_context_value in varchar2);
```

例 :

```
pay_balance_pkg.set_context ('TAX_UNIT_ID', p_tax_unit_id);
```

これは、バランスに必要なすべてのコンテキスト（アサイメント処理 ID を除く）を設定するためにコールされます。コンテキスト値は、セッション全体を通して保持されます。この値は、同じコンテキスト名を持つ後続のコールによって更新されます。

---

**注意 :** コンテキスト名は大文字でも小文字でも指定できます。ルーチンはすべてのコンテキスト名を大文字に変換します。

---

**関数 : バランス値の取得（アサイメント処理モード）:**

```
function get_value (p_defined_balance_id    in number,
p_assignment_action_id   in number,
p_always_get_db_item    in boolean default false)
return number;
```

**関数 : バランス値の取得（日付モード）:**

```
function get_value (p_defined_balance_id    in number,
p_assignment_id         in number,
p_virtual_date          in date,
p_always_get_db_item   in boolean default false)
return number;
```

バランス値は、この関数によって返されます。この関数に必要なパラメータの数は、最小限に抑えられています。バランス SQL を ff_routes から構築する必要がある場合には、PL/SQL 関数によって国別仕様コードとビジネス・グループ ID が導出されます。

---

**注意 :** バランスにコンテキストとして business_group_id が使用されている場合には、set_context ルーチンを使用して設定する必要があります。

パラメータ 'p_always_get_db_item' は無視してください。このパラメータはテスト用です。この値が 'TRUE' に設定されている場合には、この関数は最新バランス値の検索もせずに、常にデータベース・アイテムからバランスを導出します。

---

## 例

この例は、管轄区域や GRE に基づいた課税をサポートするパラメータ付きバランスへのアクセス方法を示したものです（US 固有）。

UK では、裁判所命令以外は、パラメータ付きバランスは一切使用されていません。

---

**注意 :** パラメータが付いていないバランスに関しては、pay_balance_pkg.set_context のコールは必要ありません。

---

**1. コンテキストを設定します。**

```
pay_balance_pkg.set_context ('TAX_UNIT_ID',  1);
pay_balance_pkg.set_context ('JURISDICTION_CODE', '01-123-4567');
```

**2. バランス値を取出します。**

```
bal_value := pay_balance_pkg.get_value (p_def_balance_id, p_asg_action_id);
```

**3. 異なる管轄区域コードのバランスを取出しますが、税単位 ID には同じ値を使用します。**

```
pay_balance_pkg.set_context ('JURISDICTION_CODE', '99-999-1234');
bal_value := pay_balance_pkg.get_value (p_def_balance_id, p_asg_action_id);
```

# 国別バランス初期化（UK のみ）

## バランス初期化エレメント

事前定義バランスのバランス初期化を許可するためには、次に示すエレメントを 01-Jan-0001 の日付以降の全給与計算にリンクする必要があります。

- 裁判所命令バランスの設定
- NI バランス 1 の設定
- NI バランス 2 の設定
- NI 社用車バランスの設定
- 税バランスの設定
- 社用車走行距離偶数バランスの設定
- 社用車走行距離奇数バランスの設定

これらのエレメントからの入力によって、事前定義バランスの初期値が設定されます。

### サポートされている累計期間

現在、次の累計期間がサポートされています。

- ASG_PROC_YTD
- ASG_YTD
- ASG_TD_YTD
- ASG_STAT_YTD
- PER_TD_DIR_YTD
- ASG_PROC_PTD
- ASG_ITD

## 初期化を必要とする可能性のある事前定義バランス

次に示す事前定義バランスについては、会計年度の半ばに Oracle Payroll に移行する場合には、初期化する必要のある場合があります。

法定バランスについては、P35 年末還付に報告されているので、初期化する必要があります。この場合は、ASG_TD_YTD と ASG_STAT_YTD の 2 種類の累計期間が使用されます。NI カテゴリ・バランス（NI A Able、NI A 従業員、NI A 事業主、NI A 合計など）は、歩調を合わせる必要があります。例 :

- たとえば、NI A Able バランスをロードする場合には、NI A 従業員バランスもロードする必要があります。
- NI A 事業主バランスをロードする場合には、NI 事業主バランスもロードする必要があります。

同様に、課税対象給与と PAYE も歩調を合わせる必要があります。NI Y の 1 年の未払が報告されています。累積された年に NI Y を設定するか、報告された年に NI Y 前年を設定してください。

部長の場合は、ASG_TD_YTD の数字と異なる場合でも、累計期間 PER_TD_DIR_YTD のみを初期化すればすみます（すなわち、部長は会計年度の途中まで任命されていた）。裁判所命令エレメントの中には、ORIGINAL_ENTRY_ID によって特定のエレメント・エントリと関連付けられている ELEMENT_ITD 累計期間を使用するものもあります。まずエレメント・エントリを設定し、そこから pay_balance_batch_lines の ORIGINAL_ENTRY_ID として Element_entry_id を入力する必要があります。

| バランス名 | 累計期間 | 年末に還付 |
|---|---|---|
| CAO スコットランド支払 EAS | _ASG_ITD | |
| 裁判所命令（Court Order） | _ELEMENT_ITD | |
| 裁判所命令未払控除項目 | _ELEMENT_ITD | |
| 裁判所命令未払保護支払 | _ELEMENT_ITD | |
| 裁判所命令優先順位なし | _ELEMENT_ITD | |
| EAS スコットランド | _ASG_ITD | |
| EAS スコットランド支払 | _ASG_ITD | |
| GAYE | _ASG_YTD | |
| GAYE 課税 | _ASG_YTD | |
| 総支給額 | _ASG_TD_YTD | Yes |
| | _ASG_YTD | |
| NI A Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI A 従業員 | _ASG_TD_YTD | Yes |

| バランス名 | 累計期間 | 年末に還付 |
|---|---|---|
| | _PER_TD_DIR_YTD | |
| NI A 事業主 | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI A 合計 | _ASG_TD_YTD | Yes |
| NI 未払 | _ASG_TD_YTD | |
| NI B Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI B Employee | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI B Employer | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI B Total | _ASG_TD_YTD | Yes |
| NI C Able | _ASG_TD_YTD | |
| | _PER_TD_DIR_YTD | |
| NI C CO | _ASG_TD_YTD | |
| | _PER_TD_DIR_YTD | |
| NI C CO Able | _ASG_TD_YTD | |
| | _PER_TD_DIR_YTD | |
| NI C 従業員 | _ASG_TD_YTD | |
| | _PER_TD_DIR_YTD | |
| NI C 事業主 | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI C 合計 | _ASG_TD_YTD | Yes |
| NI 社用車支払 | _ASG_STAT_YTD | |
| NI 社用車支払 2 次 | _ASG_STAT_YTD | |
| NI 社用車 1 次 | _ASG_STAT_YTD | |
| NI 社用車 2 次 | _ASG_STAT_YTD | |
| NI D Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI D CO | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI D CO Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |

| バランス名 | 累計期間 | 年末に還付 |
|---|---|---|
| NI D Employee | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI D Employer | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI D Total | _ASG_TD_YTD | Yes |
| NI E Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI E CO | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI E CO Able | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI E Employee | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI E Employer | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| NI E Total | _ASG_TD_YTD | Yes |
| NI 従業員未払 | _ASG_TD_YTD | |
| NI Employer | _ASG_TD_YTD | |
| NI 事業主未払 | _ASG_TD_YTD | |
| NI Y | _ASG_STAT_YTD | Yes |
| NI Y 昨年度 | _ASG_STAT_YTD | Yes |
| NIable Pay | _ASG_TD_YTD | |
| | _PER_TD_DIR_YTD | |
| Net Pay | _ASG_TD_YTD | |
| PAYE | _ASG_TD_YTD | Yes |
| | _PER_TD_DIR_YTD | |
| PAYE 労使紛争払戻 | _ASG_TD_ITD | |
| PAYE スタータ払戻 | _ASG_TD_YTD | |
| SMP 合計 | _ASG_TD_YTD | Yes |
| SSP 合計 | _ASG_TD_YTD | Yes |
| 退職年金手当合計 | _ASG_TD_YTD | Yes |

| バランス名 | 累計期間 | 年末に還付 |
|---|---|---|
| Taxable Pay | _ASG_TD_YTD | Yes |
| Total Deductions | _ASG_TD_YTD | |
| 寡婦および親なし | _ASG_TD_YTD | Yes |

# レポートへのバランス値の組込み（UK のみ）

本項では、ユーザーによる問合せおよびレポート作成ツールのバランス値へのアクセスを可能する UK バランス関数の PL/SQL インタフェースについて説明します。

### 利点

PL/SQL 関数を使用したバランス値の取出しには、次のようないくつかの利点があります。

- この関数は、フォームまたはレポートから簡単にコールすることができます。
- この関数は、最新バランス値（存在している場合）にアクセスします。

  これによって、パフォーマンスが必然的に最適化されます。
- この関数は、ユーザー定義のビューからコールすることができます。

  これは、この関数が WNPS や WNDS セット（パッケージへの書込み禁止状態および DML への書込み禁止状態）といったプラグマ・レベルを持っているためです。

## バランス関数

この関数へのインタフェースは、柔軟性が高く、操作も簡単です。関数内のコンテキストのハードコードされる知識は最小限に抑えられるので、バランス関数は次のようにして制御されます。

- UK バランス・ユーザー・イグジットは、日付処理とアサイメント処理の 2 つのモードで機能します。

  バランス関数の場合は、PL/SQL オーバーロード機能を使用することによってインタフェースが簡略化されています。同じ関数名が使用されますが、渡されるパラメータはモードによって異なります。日付モードの場合は、この関数は渡された日付より前にアサイメント処理を使用して値を計算します。得られた値がチェックされ、その値が表すアサイメント処理と渡された日付の間に失効していないか確認されます。
- この関数は、「最短値から」という注釈を使用します。

  値を最新バランス表から取り出せる場合には、その値がその対象となります。このため、コードのパフォーマンスが最適化されます。

UK バランス値関数を直接コールするのに使用されるインタフェースは 1 つですが、PL/SQL コードは次の 3 つのパッケージに常駐しています。

- hr_dirbal
- hr_gbbal
- hr_routes

**関数 : バランス値の取得（アサイメント処理モード）:**

```
function get_balance (p_assignment_action_id     in     number,
                      p_defined_balance_id   in     number)
return number;
```

**関数 : バランス値の取得（日付モード）:**

```
function get_balance (p_assignment_id            in     number,
                      p_defined_balance_id   in     number,
                      p_effective_date       in     date)
return number;
```

バランス値は、この関数によって返されます。この関数に必要なパラメータの数は、最小限に抑えられています。

## 例

アサイメント処理 ID が 12345、事前定義バランス ID が 111 の場合 :

```
l_balance := hr_dirbal.get_balance (12345, 111);
```

これは、アサイメント処理モードのパッケージ・コールを使用してバランス値を返します。

アサイメント ID が 2、事前定義バランス ID が 111 で、任意の日付の場合 :

```
l_balance := hr_dirbal.get_balance (2,111,to_date('01/01/1998','DD/MM/YYYY'));
```

これは、日付モードのパッケージ・コールを使用してバランス値を返します。

# 「バランスの表示」の使用方法（UK のみ）

システム定義のバランス・ビュー、PAY_GB_BALANCES_ACTION_V を使用すると、アサイメント ID またはアサイメント処理 ID のいずれかにより、バランス、バランス累計期間および値を問い合せることができます。戻された行はそれぞれが 1 つのバランス累計期間および値を表します。

| 名前 | NULL? | タイプ |
|---|---|---|
| ASSIGNMENT_ID | NOT NULL | NUMBER(9) |
| ASSIGNMENT_ACTION_ID | NOT NULL | NUMBER(15) |
| BALANCE_TYPE_ID | NOT NULL | NUMBER(9) |
| BALANCE_NAME | NOT NULL | VARCHAR2(80) |
| DEFINED_BALANCE_ID | NOT NULL | NUMBER(9) |
| DATABASE_ITEM_SUFFIX | NOT NULL | VARCHAR2(30) |
| VALUE | | NUMBER |

たとえば、特定のバランス、アサイメント処理 '5268'、バランス・タイプ「総支給額」の値を取り出す場合には、次の SQL 問合せを発行します。

```
select balance_name, database_item_suffix, value
from pay_gb_balances_by_action_v
where assignment_action_id = 5268
and balance_name = 'Gross Pay'
```

関連するバランス値があるうちは、この問合せでは次のような結果が生じます。

| バランス名 | データベース・アイテムの接尾辞 | 値 |
|---|---|---|
| 総支給額 | _ASG_PROC_PTD | 4100 |
| 総支給額 | _ASG_RUN | 4100 |
| 総支給額 | _ASG_TD_YTD | 16700 |
| 総支給額 | _ASG_YTD | 16700 |
| 総支給額 | _ASG_ITD | 16700 |

データベース・アイテム接尾辞の列が特定バランスの関連累計期間と関連があることがわかるでしょう。

このビューを使用して、特定のアサイメントに関連するバランスを見つけることができます。その場合は、次の SQL 問合せを使用します。

```
Using an example assignment ID of 662;
select distinct balance_name
from pay_gb_balances_by_action_v
where assignment_id = 662
```

これにより、次のようなフォーマットの結果が生じます。

```
BALANCE_NAME
--------------------
Attachable
Gross Pay
NI B Able
NI B Employee
NI B Employer
NI B Total
NI D Able
NI D CO
NI D CO Able
NI D Employee
NI D Employer

BALANCE_NAME
--------------------
NI D Total
NI E Able
NI E CO
NI E CO Able
NI E Employee
NI E Employer
NI E Total
NI Employer
NIable Pay
NW Earns
Net Pay
```

```
BALANCE_NAME
--------------------
PAYE
Taxable Pay
Total Deductions
Total Pay
```

**注意：** アサイメントに関連するバランスのみを示しています。

# 8

# 給与明細レポート（UK のみ）

## 給与明細レポート（UK のみ）

給与明細レポートは、印刷済の書簡紙に給与明細書を出力します。効率上、また複雑さを軽減するため、レポートの一部はビューをベースとしています。

複合動的ソートがこのレポートの特徴の 1 つです。

## パラメータ値

ユーザーがレポートを生成するために入力することのできるパラメータは次のとおりです。

| | |
|---|---|
| p_payroll_id | Number（9） |
| p_time_period_id | Number（9） |
| p_pay_advice_date | Date |
| p_assignment_id | Number（20） |
| p_bus_grp_id | Number（15） |
| p_sort_order1/6 | Char（60） |
| p_sort_order7 | Char（20） |

## 問合せ

レポートには次の 5 種類の問合せがあります。

| | |
|---|---|
| Q_Personel | 下記ビューに基づく駆動問合せです。<br>PAY_ASSIGNMENT_ACTIONS_V2. |
| Q_Payment | この問合せは、下記ビューに基づきます。<br>PAY_ELEMENT_TYPES_V1.assignment_action_id を使用して<br>メイン問合せにリンクします。 |
| Q_Deduction | この問合せは、下記ビューに基づきます。<br>PAY_ELEMENTS_TYPES_V1.assignment_action_id を使用して<br>メイン問合せにリンクします。 |
| Q_Accounts | この問合せは、下記ビューに基づきます。<br>PAY_EXTERNAL_ACCOUNTS_V<br>assignment_action_id を使用して<br>メイン問合せにリンクします。 |
| Q_Messaes | この問合せは、<br>PAY_PAYROLL_ACTIONS 表に基づきます。<br>run_payroll_action_id を使用して<br>メイン問合せにリンクします。 |

## グループ

人事グループの Formula タイプ・フィールドを次に示します。

| | |
|---|---|
| c_name | 個人の肩書き、イニシャルおよび姓を連結します。 |
| c_get_address | expense_check_send_to_address フィールドの値に従って、<br>自宅または勤務先住所を取り出します。住所 ID はパッケージ<br>pay_gb_payroll_actions.get_home_address または<br>.get_work_add から入手されます。 |
| c_ff | パッケージ<br>pay_gb_payroll_actions_pkg.get_report_db_items および<br>.get_report_balances から税詳細を入手します。 |

## トリガー

次のトリガーが使用されます。

| | |
|---|---|
| Before Report | このトリガーは、セッション ID と日付を fnd_session 表に設定します。 |
| After Form | このトリガーは、パラメータがデフォルト値のままかどうかをチェックします。そうである場合には、NULL に設定します。これは、動的ソート順の一部です。 |

## レイアウト

レイアウトは、様々なグループや要約をまとめたものです。グループの中にはオーバーフローするものが出る可能性があります。その場合には、追加ページを作成します。追加ページにはすべてのフィールドが出力されるわけではありません（個人名などのフィールドだけです）。デザイナでレポートを実行した場合には、すべてのフィールドが正しく整列されるわけではありません。しかし、correct_prt ファイルを使用して正しい環境で実行した場合には、正しく整列します。

## 動的ソート順

ユーザーは、レポートのソート順を動的に変更できます。最後のパラメータ（sort_order7）は必須で、従業員の姓またはアサイメント番号から構成されます。

ソートの作用を次に示します。

1. ソートは、最初の 6 番目のパラメータによって Not Sort（ソートしない）にデフォルト設定されているので、姓別によるソートのみ行われます。
2. ユーザーは、レポートの実行時に、必須パラメータ（Segment1 や Segment2 など）を入力できます。
3. 「フォーム後」トリガーは、ユーザーがパラメータを入力したかどうかをチェックします。そうでない場合は、パラメータのデフォルトは失なわれ、パラメータは NULL に設定されます。
4. メイン問合せが開始し、expense_check_send_to_address フィールドがチェックされます。このフィールドが設定されていれば、給与明細の住所指定に個人の自宅住所が使用され、動的ソート順は使用されません。設定されていない場合は、給与明細の順序付けに、動的ソート順が使用されます。

**警告：** ソート順がこのように設計されているため、ユーザーがメイン問合せを変更した場合には必ず、コンパイル・エラーが発生します。これは、問合せの字句パラメータにデフォルトのソート順値 segmentx が使用されているためです。問合せを変更する場合には、まず segmentx を segment1 などに変更する必要があります。これで問合せはコンパイルします。トリガーを機能させるには、segmentx に変更し直します。

# 9

# FastFormula

## FastFormula アプリケーション・ディクショナリ

FastFormula アプリケーション・ディクショナリは、FastFormula ユーザーにアプリケーション・データベースの複雑さを感じさせないようにすることを目的としています。Formula を書くときには、データベース・アイテムを参照します。このディクショナリには、FastFormula でこうしたデータベース・アイテムを抽出する SQL および PL/SQL エラー・チェック・コードを生成するのに必要な情報が含まれています。

たとえば、Formula で、EMPLOYEE_LAST_NAME というデータベース・アイテムを参照することもできます。Formula を実行すると、FastFormula はこのディクショナリ内の情報を使用して、データベースからその名称を抽出するための完全な SELECT 文を作成します。

通常は、ディクショナリの内容を認識している必要はありません。たとえば、新規エレメントを定義すると、データベース・アイテムがいくつか自動的に生成されます。FastFormula によるこうした新しいアイテムの抽出を可能にするための情報も同時に生成されます。

ただし、新しいデータベース・アイテムをディクショナリに直接定義する必要がある場合には、関連情報もロードする必要があります。次項では、ディクショナリ内に作成する必要のあるエンティティについて説明します。さらにその次の項では、新しいデータベース・アイテムを定義するための詳細な手順について説明します。

## ディクショナリ内のエンティティ

FastFormula が PER_PEOPLE という表の EMPLOYEE_LAST_NAME というデータベース・アイテムを参照する Formula を実行すると仮定します。この場合、EMPLOYEE_LAST_NAME を抽出するのに必要な SQL は次のようになります。

```
SELECT TARGET.last_name
FROM per_people              TARGET
,    per_assignment          ASSIGN
WHERE TARGET.person_id     = ASSIGN.person_id
AND ASSIGN.assignment_id   = &B1
```

この項では、こうした情報がディクショナリのどこに格納されるか、FastFormula はこうした情報をどのように組み立てて SQL 文を作成するのかということについて説明します。

ディクショナリには物理レベルの情報が格納されているということに注意してください。つまり、ディクショナリに格納されているのは SQL 文のテキストの断片であり、FastFormula はそれらの断片をつなぎ合せて完全な文を構築します。ディクショナリには、エンティティや関連についての情報は含まれていません。

## データベース・アイテムとユーザー・エンティティ

EMPLOYEE_LAST_NAME は、ディクショナリ内の FF_DATABASE_ITEMS という表の USER_NAME 列にある値です。EMPLOYEE_LAST_NAME を変数とする Formula を実行する場合には、FastFormula はこの表にアクセスします。それには次の 2 つの理由があります。

- DEFINITION_TEXT 列の値を読み取る。これは SQL の SELECT 句に割り当てられる値です。この例では、PER_PEOPLE.LAST_NAME です（TARGET は PER_PEOPLE の別名です）。
- そのデータベース・アイテムが属しているユーザー・エンティティを識別する。ユーザー・エンティティは、同じルートでアクセスできる 1 つ以上のデータベース・アイテムのグループです。例では、EMPLOYEE_DETAILS がユーザー・エンティティに当たると考えられます。

## ルートとルート・パラメータ

FastFormula は、ユーザー・エンティティ ID を使用して FF_USER_ENTITIES 表をチェックして、そのユーザー・エンティティに関連するルートを識別します。ルートは、FROM キーワードの後ろの SQL 文のテキストです。これは FF_ROUTES 表に入れられます。例では、ルートは次のとおりです。

```
per_people                 TARGET,
per_assignments            ASSIGN
WHERE TARGET.person_id   = ASSIGN.person_id
AND ASSIGN.assignment_id = &B1
```

いくつかのユーザー・エンティティが同一のルートを使用する場合、ルートにはフォーム &U#（# は順序番号）のプレースホルダが 1 つ以上含まれています。各プレースホルダは FF_ROUTE_PARAMETERS 表内のパラメータを参照します。FastFormula はこの表からパラメータ ID を識別します。

パラメータの値はユーザー・エンティティによって異なります。FastFormula は、パラメータ ID を使用して、FF_ROUTE_PARAMETER_VALUES 表内の関連するユーザー・エンティティのパラメータの値にアクセスします。各ユーザー・エンティティは個別のパラメータ値セットを持つため、ルートのテキストもユーザー・エンティティによって異なります。

例では、ルートを使用しているのが 1 つのユーザー・エンティティのみなので、ルート・パラメータはありません。

## コンテキストとルートによるコンテキスト使用

ルートにフォーム &B#（# は順序番号）の別のタイプのプレースホルダが含まれている場合もあります。これらのプレースホルダは、FF_ROUTE_CONTEXT_USAGES 表内のコンテキストを参照します。FastFormula は、この表からコンテキストの ID を識別した後で、FF_CONTEXTS 表からコンテキストの名称を識別します。コンテキストは FF_CONTEXTS 内にあらかじめ定義されていますが、絶対に変更しないでください。例として、給与 ID、組織 ID、対象日が挙げられます。

コンテキストの値は固定ではありません。この値は実行時に Formula を通じて渡されます。

例では、ルートによって 1 つのコンテキスト、すなわちアサイメント ID が要求されています。

## Formula タイプと Formula タイプによるコンテキスト使用

Formula を定義する場合は、Formula に給与計算 Formula や QuickPaint Formula などの Formula タイプを割り当てます。Formula のタイプによって、値を割り当てるコンテキストが決まります。これは、FF_FTYPE_CONTEXT_USAGES 表に定義されています。

たとえば、QuickPaint Formula は、アサイメント ID と対象日の 2 つのコンテキストに値を割り当てます。したがって、QuickPaint Formula を定義する場合には、アサイメント ID と対象日の 2 つのコンテキストを必要とするデータベース・アイテムを使用できます。ただし、ルート内の他のコンテキストを使用するデータベース・アイテムは一切使用できません。これらは値リストには表示されません。

これは、Formula に使用するデータベース・アイテムを制限するための方法の 1 つです。Formula のコンテキストに適したデータベース・アイテムしか使用できません。

データベース・アイテムがコンテキストを一切必要としないルートをベースとしている場合（たとえば、DUAL の SELECT）は、システム内のあらゆる Formula タイプがそのデータベース・アイテムにアクセスできます。

## FastFormula によるディクショナリの使用方法についての概要

1. FastFormula は、FF_DATABASE ITEMS の DEFINITION_TEXT 列の値を読み出して、SQL の SELECT 句に割り当てます。
2. 次に、FF_DATABASE ITEMS からユーザー・エンティティ ID を読み出し、それを使用して FF_USER_ENTITIES からルート ID を読み出します。
3. さらに、読み出したルート ID を使用して FF_ROUTES からルート・テキストを読み出し、SQL の FROM 句に割り当てます。
4. ルートにフォーム &U# のプレースホルダが含まれている場合には、FastFormula は FF_ROUTE_PARAMETERS にアクセスして、パラメータ ID を識別します。さらに、そのパラメータ ID を使用して、FF_ROUTE_PARAMETER_VALUES 表内の関連するユーザー・エンティティのパラメータ値を読み取ります。

5. ルートにフォーム &B# のプレースホルダが含まれている場合には、FastFormula は FF_ROUTE_CONTEXT_USAGES にアクセスして、コンテキスト ID を識別します。さらに、そのコンテキスト ID を使用して、FF_CONTEXTS 表内のそのコンテキストの名称を読み取ります。これは、Formula によって値が渡されるコンテキストのうちの 1 つであるはずです（FF_FTYPE_CONTEXT_USAGES 表内の Formula タイプによって決まります）。

# 新規データベース・アイテムの定義

新規アイテムを定義する前に、次の事項を考慮する必要があります。

- データベース・アイテムの使用をどのビジネス・グループと国別仕様に認めるか？
- データベース・アイテムを NULL 値を取ることができるものにするか、できないものにするか？

## データベース・アイテムの可用性

ビジネス・グループ ID と国別仕様コードの 2 つの属性が各ユーザー・エンティティに関連付けられます。これらの属性によって、そのユーザー・エンティティに属するデータベース・アイテムの可用性が決まります。ビジネス・グループ ID を特定の値に設定した場合には、そのビジネス・グループのもとで動作する Formula のみがそのデータベース・アイテムにアクセスできます。ビジネス・グループ ID を NULL に設定した場合には、すべてのビジネス・グループがそのデータベース・アイテムにアクセスできます。同様の原理が国別仕様コードにも当てはまります。

新規データベース・アイテムを定義する場合は、特定のビジネス・コードと国別仕様に対応付ける必要があります。コア・システムの一部として提供されている汎用の起動アイテムは、すべての Formula に使用できます。ローカライズ・グループによって国別仕様アイテムが追加されています。これらのアイテムへは、国別仕様のもとにあるあらゆるビジネス・グループがアクセスできます。

---

**注意：** データベース・アイテムの名称は、ビジネス・グループ内で一意である必要があります。

---

## NULL 条件と Not Found 条件

検証を有効にするには、FastFormula アプリケーション・ディクショナリに次の 2 つのフラグを定義する必要があります。

- NULL_ALLOWED_FLAG は、FF_DATABASE_ITEMS 表の 1 つの列であるため、各データベース・アイテムに適用されます。そのデータベース・アイテムを抽出する SQL 文が NULL 値を返す可能性がある場合には、このフラグを Yes（Y）に設定する必要があります。このフラグを No に設定しても、NULL 値が返された場合には、FastFormula はエラーを報告します。

- NOTFOUND_ALLOWED_FLAG は、FF_USER_ENTITIES 表の 1 つの列であるため、ある特定のユーザー・エンティティに属するすべてのデータベース・アイテムに適用されます。それらのデータベース・アイテムを抽出する SQL 文がいずれのアイテムの行もまったく返さない可能性がある場合には、このフラグを Yes（Y）に設定する必要があります。このフラグを No に設定しても、SQL 文が行を返さなかった場合には、FastFormula はエラーを報告します。

Formula の記述者は、これらのフラグが両方とも No に設定されていないかぎり、Formula に使用するデータベース・アイテムのデフォルトを指定する必要があります。デフォルトの詳細は、『Oracle FastFormula の使用方法』を参照してください。

## データベース・アイテムの生成手順

GRADE_RATE_USER_ENTITY というユーザー・エンティティを例にとって、データベース・アイテムの生成手順について説明します。このエンティティは、次の 3 つのデータベース・アイテムから構成されます。

- GRADE_VALUE
- GRADE_MINIMUM
- GRADE_MAXIMUM

このユーザー・エンティティは、他のユーザー・エンティティとルート（GRADE_ROUTE）を共有する場合があります。各ユーザー・エンティティはルート・パラメータ RATE_ID に一意の値を使用するので、WHERE 句はエンティティによって異なります。これらのエンティティが同一のビジネス・グループ内にある場合には、各データベース・アイテムの USER_NAME は一意である必要があります。そのための方法の 1 つが、USER_NAME にレート名を含めることです。たとえば次のようにします。
<RATE_NAME>_GRADE_VALUE.

この例では、GRADE_RATE_USER_ENTITY の RATE_ID の値を 50012 と仮定しています。わかりやすくするために、ルートの 1 つのユーザー・エンティティについてのみ検討します。

PAY_GRADE_RULES 表には、3 つのデータベース・アイテムが格納されています。これらのアイテムを抽出するのに、FastFormula は Formula によって渡されたアサイメント ID を使用します。これが Formula テキストです。

これらのデータベース・アイテムを抽出するのに必要な SQL を次に示します。

```
SELECT <DEFINITION_TEXT>
FROM   pay_grade_rules                    TARGET
,      per_assignments                    ASSIGN
WHERE  TARGET.grade_or_spinal_point_id  = ASSIGN.grade_id
AND    TARGET.rate_type                 = 'G'
AND    ASSIGN.assignment_id             = &B1
AND    TARGET.rate_id                   = &U1
```

<DEFINITION_TEXT> は、次にリストされている 3 つのデータベース・アイテムのうちの 1 つを取り得ます。

| データベース・アイテム名 | <DEFINITION_TEXT> |
|---|---|
| GRADE_VALUE | TARGET.value |
| GRADE_MINIMUM | TARGET.minimum |
| GRADE_MAXIMUM | TARGET.maximum |

次の各ステップでは、こうした情報をディクショナリにロードして、FastFormula がこの SQL を生成できるようにするための方法について説明します。こうした情報をロードするための PL/SQL の例については、この項の終わりに示されています。

### ステップ 1 SQL を記述する

SQL*Plus を使用して SQL 文を記述し、テストして、SQL 文が正しいことを確認します。FastFormula は複数の行を処理できないので、SQL 文が 2 つ以上の行を返すことがあってはなりません。

### ステップ 2 ルートをロードする

これは、PL/SQL ルーチンを使用すると上手くできます。可能な限り、主キーの順序番号値（FF_ROUTES_S.NEXTVAL など）を使用して表に挿入してください。ルートは、FF_ROUTE 表内に「long」データ型として保持されます。このため、前述の例に従って、次のようにルートを long 型の変数に割り当てることができます。

```
set escape \
DECLARE
  l_text long;
BEGIN
    l_text := '/* route for grade rates */
       pay_grade_rules                          TARGET,
       per_assignments                          ASSIGN
WHERE TARGET.grade_or_spinal_point_id        = ASSIGN.grade_id
AND TARGET.rate_type                         = ''G''
AND ASSIGN.assignment_id                     = \&B1
AND TARGET.rate_id                           = \&U1';
END;
```

最初の方に示されている元の SQL と次の点が異なっていることに注意してください。

- 「&」の前には必ずエスケープ文字があります。
- 一重引用符が 2 つの一重引用符に置き換えられています。
- 必要であれば、ルートの初めにコメントを入れることもできます。

### ステップ 3 コンテキストをロードする

次に、コンテキストを FF_ROUTE_CONTEXT_USAGES 表にロードします。この表の各列は次のようになっています。

```
名前                                      Null?    タイプ
----------------------------------------------------
ROUTE_ID                                  NOT NULL NUMBER(9)
CONTEXT_ID                                NOT NULL NUMBER(9)
SEQUENCE_NO                               NOT NULL NUMBER(9)
```

ルート ID の現在の順序番号を使用します。FF_ROUTES_S.NEXTVAL 順序を使用して FF_ROUTES 表に挿入した場合には、FF_ROUTES_S.CURRVAL になります。特定の Formula コンテキスト（例では、アサイメント ID）のコンテキスト ID は、FF_CONTEXTS 表から入手できます。順序番号は単に「B」のみです。

たとえば、ルートの 1 行を FF_ROUTE_CONTEXT_USAGES 表に挿入します（本例の PL/SQL については、この項の終わりを参照してください）。

### ステップ 4 行をユーザー・エンティティ表に挿入する

ルートごとに、少なくとも 1 行を FF_USER_ENTITIES 表に挿入します。この表には、ビジネス・グループ ID、国別仕様コード、ROUTE_ID、NOTFOUND_ALLOWED_FLAG が保持されています。

### ステップ 5 ルート・パラメータの行を挿入する

ルート内の各フォーム &U# のプレースホルダに関して、次の 2 つの表に 1 つの行を挿入する必要があります。

- FF_ROUTE_PARAMETERS。ルートを参照します。
- FF_ROUTE_PARAMETER_VALUES。ルート・パラメータの実効値が含まれており、ユーザー・エンティティを参照します。

これらの表の各列は次のようになっています。

```
SQL> desc ff_route_parameters
 名前                           Null?    タイプ
 ------------------------------ -------- ----
 ROUTE_PARAMETER_ID             NOT NULL NUMBER(9)
 ROUTE_ID                       NOT NULL NUMBER(9)
 DATA_TYPE                      NOT NULL VARCHAR2(1)
 PARAMETER_NAME                 NOT NULL VARCHAR2(80)
 SEQUENCE_NO                    NOT NULL NUMBER(9)
```

```
SQL> desc ff_route_parameter_values
 名前                           Null?    タイプ
 ------------------------------ -------- ----
 ROUTE_PARAMETER_ID             NOT NULL NUMBER(9)
 USER_ENTITY_ID                 NOT NULL NUMBER(9)
 VALUE                          NOT NULL VARCHAR2(80)
 LAST_UPDATE_DATE                        DATE
 LAST_UPDATED_BY                         NUMBER(15)
 LAST_UPDATE_LOGIN                       NUMBER(15)
 CREATED_BY                              NUMBER(15)
 CREATION_DATE                           DATE
```

FF_ROUTE_PARAMETERS に保持されるデータ型は、数値（N）かテキスト値（T）です。

例では、ルート・パラメータは RATE_ID です。GRADE_RATE_USER_ENTITY の場合、RATE_ID の値は 50012 です。例題のこれらの表に挿入する値については、この項の終わりに示されているサンプル PL/SQL を参照してください。

### ステップ 6 データベース・アイテムを挿入する

データベース・アイテムを挿入する準備ができました。この例の場合、FF_DATABASE_ITEMS 表内には同一のユーザー・エンティティを参照する行が 3 つあります。この表の各列は次のようになっています。

```
SQL> desc ff_database_items
 名前                           Null?    タイプ
 ------------------------------ -------- ----
 USER_NAME                      NOT NULL VARCHAR2(80)
 USER_ENTITY_ID                 NOT NULL NUMBER(9)
 DATA_TYPE                      NOT NULL VARCHAR2(1)
 DEFINITION_TEXT                NOT NULL VARCHAR2(240)
 NULL_ALLOWED_FLAG              NOT NULL VARCHAR2(1)
 DESCRIPTION                             VARCHAR2(240)
 LAST_UPDATE_DATE                        DATE
 LAST_UPDATED_BY                         NUMBER(15)
 LAST_UPDATE_LOGIN                       NUMBER(15)
 CREATED_BY                              NUMBER(15)
 CREATION_DATE                           DATE
```

USER_NAME はビジネス・グループ内で一意のものである必要があります。

これら 3 つの例題データベース・アイテムのこの表に挿入する値については、この項の終わりに示されているサンプル PL/SQL を参照してください。

こうしたデータベース・アイテムを作成するときには、この他の LAST_UPDATE_DATE や CREATION_DATE などの列に挿入すると便利です。

次に示す PL/SQL は、例題のデータベース・アイテムを作成します。

```
set escape \
DECLARE
  l_text                long;
  l_user_entities_seq   number;
  l_route_id            number;
    BEGIN
 --
 -- ルートをローカル変数に割り当てる
 --
    l_text := '/* route for grade rates */
    pay_grade_rules                                 TARGET,
per_assignments                                     ASSIGN
WHERE TARGET.grade_or_spinal_point_id          = ASSIGN.grade_id
AND TARGET.rate_type                           = ''G''
AND ASSIGN.assignment_id                       = \&B1
AND TARGET.rate_id                             = \&U1';
--
-- ルートを ff_routes 表に挿入する
--
insert into ff_routes
       (route_id,
       route_name,
       user_defined_flag,
       description,
       text,
       last_update_date,
       creation_date)
values (ff_routes_s.nextval,
       'GRADE_ROUTE',
       'Y',
       'Route for grade rates',
       l_text,
       sysdate,
       sysdate);
--
-- コンテキストをロードする
--
insert into ff_route_context_usages
       (route_id,
       context_id,
       sequence_no)
select ff_routes_s.currval,
       context_id,
       1
from   ff_contexts
where  context_name = 'ASSIGNMENT_ID';
--
```

```
-- ユーザー・エンティティを作成する
--
select ff_user_entities_s.nextval
into   l_user_entities_seq
from   dual;
--
select ff_routes_s.currval
into   l_route_id
from   dual;
--
insert into ff_user_entities
       (user_entity_id,
       business_group_id,
       legislation_code,
       route_id,
       notfound_allowed_flag,
       user_entity_name,
       creator_id,
       creator_type,
       entity_description,
       last_update_date,
       creation_date)
values (l_user_entities_seq,
       1,                              -- example business group id
       'GB',                           -- example legislation
       l_route_id,
       'Y',
       'GRADE_RATE_USER_ENTITY',
       50012,                          -- example creator id
       'CUST',
       'Entity for the Grade Rates',
       sysdate,
       sysdate);
--
-- ルート・パラメータを挿入する
--
insert into ff_route_parameters
       (route_parameter_id,
        route_id,
        data_type,
        parameter_name,
        sequence_no)
select  ff_route_parameters_s.nextval,
        l_route_id,
      'N',
      'Grade Rate ID',
      1
```

```
from dual;
--

insert into ff_route_parameter_values
       (route_parameter_id,
       user_entity_id,
       value,
       last_update_date,
       creation_date)
select ff_route_parameters_s.currval,
       l_user_entities_seq,
       50012,
       sysdate,
       sysdate
from   dual;
--
-- 3 つのデータベース・アイテムを挿入する
--
insert into ff_database_items
       (user_name,
       user_entity_id,
       data_type,
       definition_text,
       null_allowed_flag,
       description,
       last_update_date,
       creation_date)
values ('GRADE_VALUE',
       l_user_entities_seq,
       'T',
       'TARGET.value',
       'Y',
       'Actual value of the Grade Rate',
       sysdate,
       sysdate);
--
insert into ff_database_items
       (user_name,
       user_entity_id,
       data_type,
       definition_text,
       null_allowed_flag,
       description,
       last_update_date,
       creation_date)
values ('GRADE_MINIMUM',
       l_user_entities_seq,
```

```
        'T',
        'TARGET.minimum',
        'Y',
        'Minimum value of the Grade Rate',


        sysdate,
        sysdate);
--
insert into ff_database_items
        (user_name,
        user_entity_id,
        data_type,
        definition_text,
        null_allowed_flag,
        description,
        last_update_date,
        creation_date)
values ('GRADE_MAXIMUM',
        l_user_entities_seq,
        'T',
        'TARGET.maximum',
        'Y',
        'Maximum value of the Grade Rate',
        sysdate,
        sysdate);

END;
/
```

# PL/SQL からの FastFormula のコール

Oracle FastFormula は、プロフェッショナル・ユーザーに対して使用しやすいツールを提供します。ユーザーは簡単なコマンドと構文を使用することにより、独自の検証ルールや給与計算を作成できます。

Formula のコールや出力処理の実行エンジンは、リリース 11 までは Oracle Human Resources および Oracle Payroll 製品内に隠されていました。PL/SQL をコールする当初のエンジンは Pro*C で作成されていました。これは複雑なものであり、ユーザー・イグジットから、または別の C インタフェースから直接にコールすることしかできませんでした。

今では、ユーザーがフォーム、レポートまたはその他の PL/SQL パッケージから Formula を直接にコールできるような、新しい実行エンジン、つまりインタフェースが提供されています。このインタフェースにより、ユーザーは既存の検証または給与計算 Formula をコールして、オンラインまたはバッチ処理に含めることができます。また、他のタイプの検証または計算についても、ユーザーは独自の Formula を定義してコールすることができます。ユーザーは FastFormula を使用することにより、オラクル人事管理システムのデータベース・アイテム（DBI）および機能に自動的にアクセスし、デートトラックされる計算およびビジネスのルールを自動的に使用できることになります。

FastFormula の基本概念は以前と変わりありません。

入力 -> 処理 -> 出力

ユーザーはどのような入力を提供し、Formula から作成される出力をどのように処理するかを自由に決定できますが、そのためには、入力と出力の両方を処理する呼出しコードを作成する必要があります。

この章では、PL/SQL から FastFormula をコールするための概要および技術情報を提供します。PL/SQL コーディング技術と Oracle FastFormula の知識が必要ですが、実行エンジンの内部機構について理解している必要はありません。

## 実行エンジン・インタフェース

FastFormula の実行エンジンに対して 2 つのインタフェースがあります。

- サーバー側

  バッチ・プロセスまたはサーバー上で実行する Formula の場合は、このインタフェースを使用します。参照 : サーバー側インタフェース : 9-15 ページ

- クライアント側

  このインタフェースは、Formula を即時に実行するために、フォームおよびレポートからの直接コールが必要な場合にのみ使用します。クライアントからサーバー・エンジンをコールするためのカスタムのラッパー・パッケージを作成することもできます。参照 : クライアント側のコール・インタフェース : 9-20 ページ

**注意：** Oracle tools の中には現在 PL/SQL V1.x のみが使用されるものがあります。このバージョンは、サーバー・インタフェースで必要なレコード表データ構造をサポートしていません。クライアント側のバージョンは、この現在の制約を解決した上で作成されています。

## ファイルの保管場所

実行エンジンのファイルは **$FF_TOP/admin/sql** に保管されています。

- ffexec.pkh および ffexec.pkb

  サーバー側の実行エンジン・パッケージのヘッダーおよび本体ファイル。

- ffcxeng.pkh および ffcxeng.pkb

  クライアント側バージョンの実行エンジン・パッケージのヘッダーおよび本体ファイル。

**注意：** 特にフォーム・クライアント用に用意された特別なインタフェースが *ff_client_engine* モジュールに入っています。このインタフェースでは、多量のネットワーク・コールによりオーバーヘッドが発生することを避けるために、固定数のパラメータが使用されます。参照 : 特別なフォーム・コール・インタフェース : 9-24 ページ

## Formula のデートトラック

オラクル人事管理システム製品内のすべての Formula はデートトラックされるため、デートトラックを使用して、検証ルールまたは計算の変更履歴を維持できます。

システムは実行エンジンに対するシステム定義のインタフェースを通じて、有効日の設定または変更を自動的に管理します。ユーザーが独自の Formula を実行するときには、ユーザーはセッションの有効日の設定も管理する必要があります。つまり、実行エンジン・インタフェースをコールする前に、FND_SESSIONS 表の中に行を挿入する必要のある場合があります。FND_SESSIONS の中に現行の SQL*PLUS の session_id がない場合、または実行される Formula が、デートトラックされている表を参照するデータベース・アイテムにアクセスする場合には、この作業が必要になります。

**注意：** Formula を実行するときには必ず有効日を確認してください。この日付はデータベース・アイテムの値や、Formula に含めるすべての関数に影響を与えます。

# R11i での変更点

## サーバー側インタフェースとクライアント側インタフェース

R11i では、クライアント側インタフェースには下位互換性があります。R11i で使用されるクライアント側の PL/SQL 環境は、サーバー側インタフェースを使用できます。

## NUMBER および DATE の入力と出力

入力パラメータは正しい形式の文字列で渡す必要があります。R11i では、FND_NUMBER.NUMBER_TO_CANONICAL ルーチンを使用して NUMBER の入力を書式化します。DATE の入力を書式化するには FND_DATE.DATE_TO_CANONICAL を使用します。

出力値は、前述のように書式化された文字列として戻されます。NUMBER 出力を NUMBER 値に変換するには、FND_NUMBER.CANONICAL_TO_NUMBER ルーチンを使用します。DATE 出力を DATE 値に変換するには、FND_DATE.CANONICAL_TO_DATE を使用します。

フォーム・コードの場合には、APP_NUMBER および APP_DATE パッケージの中の対応するルーチンを使用することにより、パフォーマンスが向上する場合があります。

この一連の変更は、FastFormula 実行エンジンへのすべてのインタフェースに適用されます。

## DATE_EARNED コンテキストと BALANCE_DATE コンテキスト

R11i では、DATE_EARNED および BALANCE_DATE コンテキストのデータ型は DATE です。R11i 以前は、これらのコンテキストのデータ型は TEXT でした。

# サーバー側インタフェース

この項では、サーバー実行エンジンへのインタフェースについて、そして他の PL/SQL からこのモジュールをコールする方法について説明します。

このバージョンのインタフェースをお薦めします。このバージョンでは、ネットワークの要求を比較的低くおさえて、最大限の柔軟性が提供されています。しかし、このバージョンではレコード表データ構造のサポートが必要であるため、PL/SQL V2.3 以上でしか使用できません。

## ユーザー・データ構造

サーバー側インタフェースを使用するときには、2 つの重要なユーザー・データ構造があります。入力表と出力表です。

**表 9-1 入力表**

| 入力表 | |
|---|---|
| NAME | RATE または ASSIGNMENT_ID などの入力名 |
| DATATYPE | DATE、NUMBER または TEXT |
| CLASS | 入力のタイプ : CONTEXT または INPUT<br>Formula を正しくコールするために入力がコンテキストであるか通常の入力パラメータであるかを知る必要はないため、このフィールドは必須ではありません。 |
| VALUE | コンテキストまたは入力として Formula に渡される実際の値。<br>このフィールドのタイプは varchar2(240) です。つまり、NUMBER および DATE データ型の場合には、正しい書式で値を渡す必要があります。この仕組みについては、コード・サンプルを参照してください。 |

**表 9-2 出力表**

| 出力表 | |
|---|---|
| NAME | RESULT1 または MESSAGE などの出力名 |
| DATATYPE | DATE、NUMBER または TEXT |
| VALUE | Formula から戻される実際の値 |

**注意 :** すべての入力および出力の名称は大文字にする必要があります。また、たとえば入力パラメータが Formula からの戻り値と同じであるなど、入力表と出力表の両方で同じ名称を使用できます。しかし、CONTEXT は入力表でしか使用できません。

入力表と出力表は両方ともに、*ff_exec.init_formula* プロシージャをコールすることにより初期化された後、Formula の実行に必要なコンテキストを含めてすべての入力、そして戻す必要のあるすべての出力を含むものになります。

初期化と実行コールの間この表を維持するのはユーザーの責任です。

---

**注意：** これらの表の索引値は正の値になりますが、1 から開始するとはかぎりません。これらの表でアクセスおよびループを行う場合には、必ず「最初」および「最後」の表属性を使用してください。参照 : 例 : 9-18 ページ

---

## 使用可能なコール

次のプロシージャ・コールを使用できます。ここでは、プロシージャに渡すことができるパラメータの詳細もあわせて説明しています。

---

**注意：** パラメータのクラス（IN、OUT または IN/OUT）については、適切なパッケージ・ヘッダーを参照してください。

---

### プロシージャ： init_formula

このコールは、特定の Formula 用に実行エンジンを初期化します。つまり、Formula を実行することをエンジンに対して宣言します。Formula を実行する前にこのプロシージャをコールする必要がありますが、その後は初期化プロシージャを再度コールすることなく、必要な回数のみその Formula を実行できます。これについては後の例で理解できるはずです。

**表 9-3 init_formula のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_formula_id | number | 実行する formula_id |
| p_effective_date | date | 実行の有効日 |
| p_inputs | ff_exec.inputs_t | 入力変数情報 |
| p_outputs | ff_exec.outputs_t | 出力変数情報 |

### プロシージャ： run_formula

このコールは、指定された入力を受け取り、Formula からの結果を戻して、実際に Formula を実行します。このプロシージャを使用する前に、init_formula プロシージャがコールされている必要があります（例を参照）。

**表 9-4 run_formula のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_inputs | ff_exec.inputs_t | Formula への入力 |
| p_outputs | ff_exec.outputs_t | Formula からの出力 |
| p_use_dbi_cache | boolean | TRUE の場合、データベース・アイテム・キャッシュは実行中アクティブになる。そうでない場合はアクティブにならない。デフォルトは TRUE |

## 追加のコメント

Formula に入力や出力がない場合には、p_inputs および p_outputs パラメータを NULL にすることもできます（後者はあまり考えられない状況ですが）。

p_use_dbi_cache は、データベース・アイテム値のキャッシングを無効にする必要があるような、特別な状況でのみ FALSE に設定されます。このような指定は、フェッチされたデータベース・アイテムの値を無効とするようなコードからエンジンがコールされた場合に必要となることがあります。

たとえば、データベース・アイテム ASG_STATUS がビジネス・ルール検証に使用される Formula 内からアクセスされ、その後これを使用してアサイメントのステータスが変更される場合には、その後の Formula でこのデータベース・アイテムの読込みを行うことを考慮して、データベース・アイテム・キャッシュを無効にすることが考えられます。

## 例

この例では、次の Formula を実行することを想定しています。DATABASE_ITEM には ASSIGNMENT_ID コンテキストが必要なことに注意してください。

この Formula 自体は別に意味のないものです。説明のためにのみ使用しています。

```
inputs are input1, input2 (date), input3 (text)
dbi = DATABASE_ITEM
ret1 = input1 * 2
return ret1, input2, input3
```

次の PL/SQL の無名ブロックを使用して Formula を実行できます。ここでは、Formula を一度のみ初期化すれば何度でも実行できることを示すために、何回かコールを行っています。

```
declare
  l_input1         number;
  l_input2         date;
  l_input3         varchar2(80);
  l_assignment_id number;
  l_formula_id     number;
  l_effective_date date;
  l_inputs         ff_exec.inputs_t;
  l_outputs        ff_exec.outputs_t;
  l_loop_cnt       number;
  l_in_cnt         number;
  l_out_cnt        number;
begin
  -- Set up some the values we will need to exec formula.
  l_formula_id     := 100;
  l_effective_date := to_date('06-05-1997', 'DD-MM-YYYY');
  l_input1         := 1000.1;
  l_input2         := to_date('01-01-1990', 'dd-mm-yyyy');
  l_input3         := 'INPUT TEXT';
  l_assignment_id := 400;
  -- Insert FND_SESSIONS row.
  insert into fnd_sessions (
 session_id,
 effective_date)
  values (userenv(`sessionid'),
 l_effective_date);
  -- Initialise the formula.
  ff_exec.init_formula(l_formula_id, l_effective_date, l_inputs,
l_outputs);
  -- We are now in a position to execute the formula.
  -- Notice that we are illustrating here that the formula can
  -- be executed a number of times, in this case setting a new
  -- input value for input1 each time.
  for l_loop_cnt in 1..10 loop
    -- The input and output table have been initialized.  We now have
    -- to set up the values for the inputs required.  This includes
    -- those for the 'inputs are' statement and any contexts.
    for l_in_cnt in l_inputs.first..l_inputs.last loop
      if(l_inputs(l_in_cnt).name = 'INPUT1') then
        -- Deal with input1 value.
        l_inputs(l_in_cnt).value := fnd_number.number_to_canonical(l_input1);
      elsif(l_inputs(l_in_cnt).name = 'INPUT2') then
        -- Deal with input2 value.
        l_inputs(l_in_cnt).value := fnd_date.date_to_canonical(l_input2);
      elsif(l_inputs(l_in_cnt).name = 'INPUT3') then
        -- Deal with input3 value.
        l_inputs(l_in_cnt).value := l_input3;
```

```
        -- no conversion required.
      elsif(l_inputs(l_in_cnt).name = 'ASSIGNMENT_ID') then
        -- Deal with the ASSIGNMENT_ID context value.
        l_inputs(l_in_cnt).value := l_assignment_id;
      end if;
    end loop;
    ff_exec.run_formula(l_inputs, l_outputs);
    -- Now we have executed the formula. We are able
    -- to display the results.
    for l_out_cnt in l_outputs.first..l_outputs.last loop
      hr_utility.trace('output name     : ' || l_outputs(l_out_cnt).name);
      hr_utility.trace('output datatype : ' || l_outputs(l_out_cnt).datatype);
      hr_utility.trace('output value    : ' || l_outputs(l_out_cnt).value);
    end loop;
  end loop;
  -- We can now continue to call as many formulas as we like,
  -- always remembering to begin with a ff_exec.init_formula call.
  -- Note: There is no procedure to be called to
  -- shut down the execution engine.
end;
/
```

以前に説明したように、適切なバージョンの PL/SQL を実行していないクライアントから実行エンジンをコールする場合には、エンジンのコールを処理するパッケージを作成するか、次に説明するように、クライアント・エンジンをコールすることを考えてみる必要があります。

# クライアント側のコール・インタフェース

この項では、ユーザーの観点からクライアント実行エンジンへのインタフェースの詳細を説明するとともに、その他の PL/SQL からモジュールをコールする方法を説明します。

> **注意：** これらクライアント側のコールは、フォームからプロシージャがコールされたときに問題を発生させる、オーバーロードの使用を回避するように設計されています。

## このインタフェースを使用する状況とは

このインタフェースは、クライアント上の PL/SQL のバージョンが V2.3 よりも前の（レコード表をサポートしない）ものである場合に使用します。これはおそらく、最も使用しやすいインタフェースです。しかし、多量のネットワーク往復処理が発生するため、高いパフォーマンスが必要な状況ではお薦めできません。このような場合には、特別のフォーム・インタフェースを使用することを考えてみてください。

## ユーザー・データ構造

クライアント側のコールには、ユーザーから見えるデータ構造はありません。

## 使用可能なコール

次のプロシージャ・コールを使用できます。ここでは、プロシージャに渡すことができるパラメータの詳細もあわせて説明しています。

> **注意：** パラメータのクラス（IN、OUT または IN/OUT）については、適切なパッケージ・ヘッダーを参照してください。

### プロシージャ： init_formula

このコールは、特定の Formula 用に実行エンジンを初期化します。つまり、Formula を実行することをエンジンに対して宣言します。Formula を実行する前にこのプロシージャをコールする必要がありますが、その後は初期化プロシージャを再度コールすることなく、必要な回数のみその Formula を実行できます。これについては後の例で理解できるはずです。

**表 9-5 init_formula のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_formula_id | number | 実行する Formula_id |
| p_effective_date | date | 有効実行日 |

### プロシージャ： set_input

このコールは、Formula への入力の値を設定します。FastFormula が処理できる多様なデータ型に対応して、適切な文字列に値を変換する必要があります。

**表 9-6 set_input のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_input_name | varchar2 | 設定する入力の名称 |
| p_value | varchar2 | 設定する入力パラメータ |

### プロシージャ： run_formula

このコールは、指定された入力を受け取り、Formula からの結果を戻して、実際に Formula を実行します。このプロシージャを使用する前に、init_formula プロシージャがコールされている必要があります（例を参照）。

run_formula にはパラメータはありません。

### プロシージャ： get_output

このコールは、Formula から戻される出力値を受け取ります。FastFormula が処理できる多様なデータ型に対応して、出力を適切に変換する必要があります。

**表 9-7 get_output のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_input_name | varchar2 | 設定する入力の名称 |
| p_return_value | varchar2 | varchar2 出力の値 |

## 例

次の例は、前に使用したものと同じ Formula を使用しています。

```
inputs are input1, input2 (date), input3 (text)
dbi = DATABASE_ITEM
ret1 = input1 * 2
return ret1, input2, input3
```

次の PL/SQL の無名ブロックを使用して Formula を実行できます。

```
declare
  l_input1         number;
  l_input2         date;
  l_input3         varchar2(80);
  l_output1        number;
  l_output2        varchar2(12);
  l_output3        varchar2(80);
  l_assignment_id number;
  l_formula_id     number;
  l_effective_date date;
  l_loop_cnt       number;
begin
-- Set up the values we need to execute the formula.
  l_formula_id     := 100;
  l_effective_date := to_date('06-05-1997', 'DD-MM-YYYY');
  l_input1         := 1000.1;
  l_input2         := to_date('01-01-1990', 'dd-mm-yyyy');
  l_input3         := 'INPUT TEXT';
  l_assignment_id := 400;
-- Insert FND_SESSIONS row.
insert into fnd_sessions (
        session_id,
        effective_date)
values (userenv(`sessionid'),
        l_effective_date);
```

```
-- Initialize the formula. ff_client_engine.init_formula(l_formula_id,l_effective_
date);
-- We are not in a position to execute the formula.
-- Notice that we are illustrating here that the formula can
-- be executed a number of times, in this case setting a new
-- input value for input1 each time.
  for l_loop_cnt in 1..10 loop
-- The input and output tables have been initialized.
-- We now have to set up the values for the inputs required.
-- This includes those for the 'inputs are' statement
-- and any contexts.
-- Note how the user has to know the number of inputs the
-- formula has.
    ff_client_engine.set_input('INPUT1', fnd_number.number_to_canonical(l_input1));
    ff_client_engine.set_input('INPUT2', fnd_date.date_to_canonical(l_input2));
    ff_client_engine.set_input('INPUT3', l_input3);
    ff_client_engine.set_input('INPUT3', l_input3);
    ff_client_engine.set_input('ASSIGNMENT_ID', l_assignment_id);
    ff_client_engine.run_formula;
-- Now we have executed the formula. Get the results.
    ff_client_engine.get_output('RET1',   l_output1);
    ff_client_engine.get_output('INPUT2', l_output2);
    ff_client_engine.get_output('INPUT3', l_output3);
-- OK. Finally, display the results.
    hr_utility.trace('RET1 value   : ' || output1);
    hr_utility.trace('INPUT2 value : ' || l_output2);
    hr_utility.trace('INPUT3 value : ' || output3)
  end loop;
-- We can now continue to call as many formulas as we like,
-- always remembering to begin with a
-- ff_client.init_formula call.
-- Note: There is no procedure to be called to
-- shut down the execution engine.
end;
/
```

# 特別なフォーム・コール・インタフェース

この項では、ユーザーの観点から特別なフォームのクライアント実行エンジン・インタフェースの詳細を説明するとともに、フォームからモジュールをコールする方法を説明します。

## このインタフェースを使用する状況とは

PL/SQL V2.3 以上をサポートしていない（PL/SQL レコード表を使用できない）フォームまたはレポート・クライアントから直接に Formula を実行する場合には、このインタフェースを使用することをお薦めします。

## ユーザー・データ構造

クライアント側のコールには、ユーザーから見えるデータ構造はありません。

## 使用可能なコール

次のプロシージャ・コールを使用できます。ここでは、プロシージャに渡すことができるパラメータの詳細もあわせて説明しています。

---

**注意：** パラメータのクラス（IN、OUT または IN/OUT）については、適切なパッケージ・ヘッダーを参照してください。

---

### プロシージャ : run_id_formula

このコールは、特定の Formula 用に実行エンジンを初期化した後、指定された入力およびコンテキスト引数を使用して、Formula を実行します。最後に、追加の引数セットを使用してユーザーに適切な結果を戻します。このため、この形式のコールではネットワークへの往復が 1 度のみですみます。この短所は、対処できる入力および戻りの数に制限があることです（これは PL/SQL V1.0 の制約によるものです）。

---

**注意：** 実行する Formula の formula_id がわかっているときには、このプロシージャ・コールを使用します。名称のみが判明しているときには、別のプロシージャ・コール（run_name_formula - 下記参照）を使用します。

---

**表 9-8 run_id_formula のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_formula_id | number | 実行する formula_id |
| p_effective_date | date | 有効実行日 |
| p_input_name01 . . . 10 | varchar2 | 入力名 01 . . . 10 |
| p_input_value01 . . . 10 | varchar2 | 入力パラメータ 01 . . . 10 |
| p_context_name01 . . . 14 | varchar2 | コンテキスト名 01 . . . 14 |
| p_context_value01 . . . 14 | varchar2 | コンテキスト値 01 . . . 14 |
| p_return_name01 . . . 10 | varchar2 | 戻り名 01 . . . 10 |
| p_return_value01 . . . 10 | varchar2 | 戻り値 01 . . . 10 |

### プロシージャ : run_name_formula

このコールは、特定の Formula 用に実行エンジンを初期化した後、指定された入力およびコンテキスト引数を使用して、Formula を実行します。最後に、追加の引数セットを使用してユーザーに適切な結果を戻します。このため、この形式のコールではネットワークへの往復が 1 度のみですみます。この短所は、対処できる入力および戻りの数に制限があることです（これは PL/SQL V1.0 の制約によるものです）。

**注意：** 実行する Formula の名称とタイプがわかっているときには、このプロシージャ・コールを使用します。ID しかわからないときには、run_id_formula コール（前述参照）を使用します。

**表 9-9 run_name_formula のパラメータ**

| パラメータ名： | データ型 | コメント |
|---|---|---|
| p_formula_type_name | number | Formula タイプ |
| p_formula_name | varchar2 | 実行する Formula の名称 |
| p_effective_date | date | 有効実行日 |
| p_input_name01 . . . 10 | varchar2 | 入力名 01 . . . 10 |
| p_input_value01 . . . 10 | varchar2 | 入力パラメータ 01 . . . 10 |
| p_context_name01 . . . 14 | varchar2 | コンテキスト名 01 . . . 14 |
| p_context_value01 . . . 14 | varchar2 | コンテキスト値 01 . . . 14 |
| p_return_name01 . . . 10 | varchar2 | 戻り名 01 . . . 10 |
| p_return_value01 . . . 10 | varchar2 | 戻り値 01 . . . 10 |

# ロギング・オプション

PL/SQL エンジンを通じて Formula を実行しようとして、問題が発生することもあります。多くの場合には、エラー・メッセージによって問題を特定することができます。しかし、さらに詳しい情報が必要になることもあります。

実行エンジンからロギング情報を出力するように設定することができます。この項では、ロギング・オプションを有効にして使用する方法を説明します。

**注意：** ロギングの出力では、Oracle Human Resources の標準トレース機能が使用されます。

## ロギング・オプションの有効化

実行エンジンのロギング・オプションを設定するには、ff_utils.set_debug プロシージャをコールします。このプロシージャには次のような定義があります。

```
procedure set_debug
(
  p_debug_level in binary_integer
);
```

オプションの数値は、それぞれが定数で表される 2 つの値の乗数であるため、適切な値が加算されます。

たとえば、ルーティングおよび DBI キャッシュ・デバッグ・オプションを設定するには（下記参照）、次のコールを使用します（SQLPLUS から）。

```
SQL> execute ff_utils.set_debug(9)
```

9 という値は（1+8）です。

定義されている定数を使用することもできます。例：

```
SQL> execute ff_utils.set_debug(ff_utils.ROUTING +
ff_exec.DBI_CACHE_DBG)
```

## FF_DEBUG プロファイル・オプション

実行エンジンをフォームからコールする場合には、FF_DEBUG プロファイル・オプションを使用してロギング・オプションを有効にすることができます。

一連の文字を使用して、設定するロギング・オプションを指定します。ユーザー・イグジット・ロギングを有効にするために、X を指定する必要があります。たとえば、プロファイル・オプションを XDR と設定すると、データベース・アイテム・キャッシュとルーティング情報が開始されます。

指定可能な文字の全リストは次のとおりです（各ロギング・オプションの説明は、「使用可能な情報の要約」を参照してください）。

**表 9-10 FF_DEBUG プロファイル・オプションの値**

| 文字 | 意味 . . |
|---|---|
| R | ff_utils.ROUTING |
| F | ff_exec.FF_DBG |
| C | ff_exec.FF_CACHE_DBG |
| D | ff_exec.DBI_CACHE_DBG |
| M | ff_exec.MRU_DBG |
| I | ff_exec.IO_TABLE_DBG |

## 使用可能な情報の要約

ここでは、設定に使用される記号および相当するバイナリ値を含めて、各ロギング・オプションについて簡単に説明します。

**注意：** これらのオプションの多くでは、出力を解釈するために、実行エンジン・コードの仕組みについてのある程度の知識が必要になります。

### ff_utils.ROUTING : 1

ルーティング。実行エンジンの実行中にアクセスされる関数およびプロシージャに関する出力情報です。目に見える出力には次のものがあります。

- In :run_formula
- Out :run_formula

### ff_exec.FF_DBG : 2

このデバッグ・レベルは、ヘッダーには定義されていますが、現在使用されていません。

### ff_exec.FF_CACHE_DBG : 4

Formula キャッシュ・デバッグ。現在実行中の Formula についての情報を、データ項目使用行を含めて表示します。

### ff_exec.DBI_CACHE_DBG : 8

データベース・アイテム・キャッシュ・デバッグ。データベース・アイテム・キャッシュに保持されているアイテムについての情報を表示します。これらのアイテムは特定の Formula に束縛されるものではありません。

**ff_exec.MRU_DBG : 16**

一番最近使用された Formula チェーン。現在 MRU チェーンに保持されている Formula についての情報を表示します。表示される情報には、表索引、formula_id、スティッキ・フラグおよび Formula 名が含まれます。

**ff_exec.IO_TABLE_DBG : 32**

入力および出力表デバッグ。現在入力および出力表に保持されているアイテムについての情報を表示します。これには、ユーザーが設定した情報と Formula エンジンが設定した情報の両方が含まれます。

## オプションの使用方法

どのような問題が発生するかを予想することはできないため、一般的なアドバイスしかできません。いくつかのヒントがあります。

ROUTING は、コードを理解している場合にのみ有用です。エラーが発生して、発生場所が特定できない場合、プロシージャをトレースすることによって問題がはっきりする場合があります。

FF_CACHE_DBG は、実行エンジンにどのような基本 Formula 情報が保持されているかを確認します。予想どおりのものであることを確認するのに便利です。

IO_TABLE_DBG は、Formula との間で実際に受け渡された内容を確認します。

# 10

# フレックスフィールド値の検証

## フレックスフィールド値の検証

Oracle Self Service Human Resources、アプリケーション・データ・エクスチェンジおよびフォームの中には、人事管理システム API を使用してデータベースの中にデータを記録するものがあります。データ・アップロード・プログラムなど、ユーザー・サイトのカスタム・プログラムの中にも API を使用するものがある場合があります。

リリース 11*i*（および R11.0 パッチ・セット D）以降は、API は値セットを使用してフレックスフィールド値を検証します（Professional Forms ユーザー・インタフェースと同じ方法）。これにより、値セット定義を 1 つの場所でインプリメントおよび保管できるという利点が生じます。以前のリリースでは、API はスケルトン・フレックスフィールド検証のサーバー側パッケージへの PL/SQL コールを使用して、フレックスフィールド値を検証していました。これらのパッケージは使用されなくなりました。

この章では、API がフレックスフィールド値セットを使用するときに発生する可能性のある問題の解決方法を説明します。これらの問題は、API 検証で自動的に使用可能にならないオブジェクトを値セットが参照する場合に発生します。

要約すれば、値セットが次のものを参照する場合に、問題が発生する可能性があります。

- ユーザー・プロファイル・オプション
- フォーム・ブロック・フィールド項目
- FND_SESSIONS データベース表の中の行

また、次のような場合にも問題が発生する可能性があります。

- コンテキスト・フィールドの値リストが不完全

この章ではこれ以降、これらの問題と推奨する解決策を詳しく説明します。これらすべての解決策における変更点は、エンドユーザーからは見えないものです。データベースの中のデータの物理的位置の変更も必要ありません。

## ユーザー・プロファイル・オプションの参照

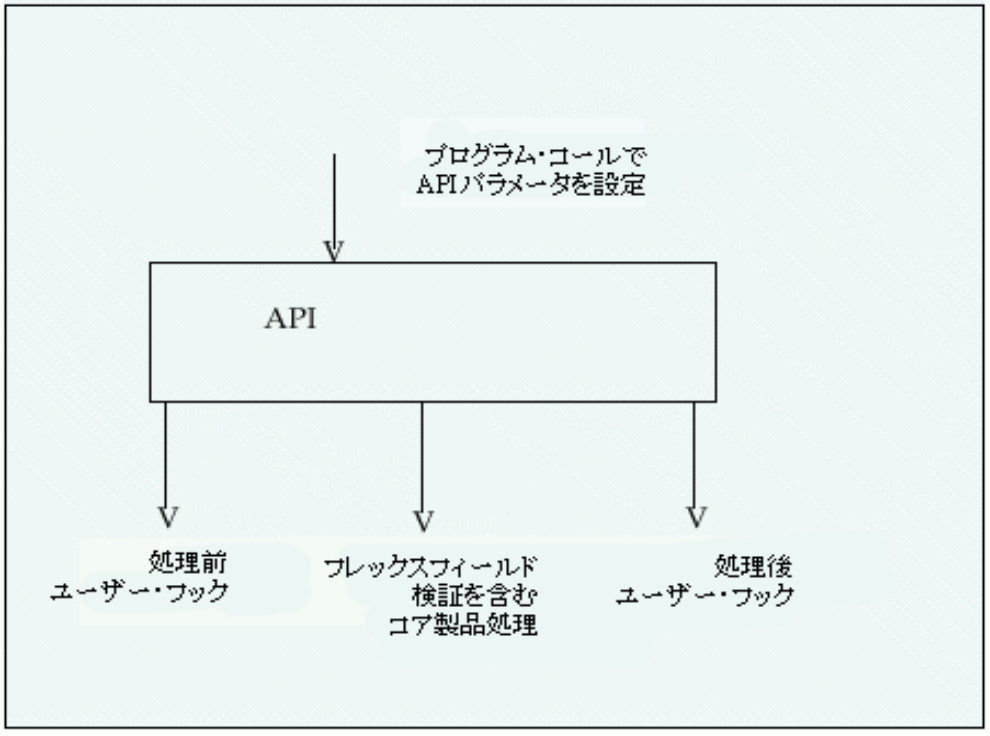
値セットの中でプロファイル・オプションを参照しても、Professional Forms ユーザー・インタフェースまたは Self Service Human Resources では問題は発生しません。ユーザーはこれらのインタフェースにログオンした場合、サイト、アプリケーション、職責またはユーザー・レベルで定義されたプロファイルを使用できます。

しかし、API が SQL*Plus データベース・セッションで直接に実行された場合には、アプリケーション・ログオンはありません。プロファイルがサイト・レベルで定義されていない場合には、この値は NULL になります。プロファイルがサイト・レベルで定義されている場合にも、適切な値であるとは限りません。たとえば、ビジネス・グループの設定について、PER_BUSINESS_GROUP_ID プロファイルがサイト・レベルで 0（ゼロ）の値に設定されています。ビジネス・グループの設定を使用しない場合、フレックスフィールド検証では行が検出されず、すべてのデータ値は無効として拒否されます。

### 推奨する解決策

フレックスフィールド検証を実行する前に、値セットの中で参照するすべてのプロファイルが適切な値に設定されていることを確認します。API ユーザー・フックを使用してこれを行うことができます。次の例は、PER_BUSINESS_GROUP_ID プロファイルを使用しています。

**図 10-1 API ユーザー・フックを使用したビジネス・グループ ID の設定**




「処理前」ユーザー・フック・コールを定義して、PER_BUSINESS_GROUP_ID プロファイルを設定します。API ユーザー・フックが必須の p_business_group_id パラメータを提供する場合は、このパラメータ値から直接にプロファイルを設定できます。そうでない場合は、API の必須の主キー・パラメータ値を使用して、最初にデータベース表から business_group_id 値を導出します。

未定義の場合、または 0（ゼロ）に設定されている場合以外は PER_BUSINESS_GROUP_ID プロファイルに値を入れないでください。プロファイルに 0（ゼロ）以外の値が定義されているときには、変更しないでください。これは、Professional ユーザー・インタフェースおよび Self Service Human Resources に影響を及ぼさないようにするためです。

「処理前」ユーザー・フック・パッケージは、実際にいつ PER_BUSINESS_GROUP_ID プロファイルを設定したかも記憶する必要があります。これは、パッケージのグローバル変数を使用して行うことができます。

解決策の 2 番目の部分は、PER_BUSINESS_GROUP_ID プロファイルを元の 0（ゼロ）または NULL 値にリセットするために、「処理後」ユーザー・フックを定義することです。これは、「処理前」プロセスが実際に値を変更した場合にのみ必要になります。これは、API が次にコールされたときに、プロファイルに正しい値が設定されるようにするためです。

API ユーザー・フックの使用方法の詳細は、「オラクル人事管理システムの API」の章を参照してください。

### その他の解決策

この問題が 1 つのプログラムでのみ発生する場合には、各 API コールの直前に PER_BUSINESS_GROUP_ID プロファイルを設定するようにプログラムを変更できます。しかし、同じ API をコールするその他のプログラムを将来使用する予定があるときには、これらのプログラムでも PER_BUSINESS_GROUP_ID プロファイルを設定することを覚えておく必要があります。

## フォーム・ブロック・フィールド項目の参照

値セットがフォーム・ブロック・フィールド項目を参照する場合、フォーム項目値をサーバー側で解決できないため、API がフレックスフィールド検証を実行するとエラーが発生します。この問題は、Oracle Self Service Human Resources および API をコールするカスタム・コードのすべてに影響を与えます。

### 推奨する解決策

この解決策は 3 つの部分に分かれます。

1. ブロック項目の参照がすべてカスタム・プロファイル名に変更されるように、値セットを変更します。プロファイルは実行時に動的に作成および設定できるため、これらのプロファイルを Oracle Applications データ・ディクショナリ内で定義する必要はありません。
2. 変更された値セットが正しく機能するためには、API がフレックスフィールド検証を実行する前に、プロファイルに値が入っている必要があります。PER_BUSINESS_GROUP_ID プロファイルの問題の場合と同じように、API「処理前」ユーザー・フックを使用してプロファイル値を設定する必要があります。必要な値の中には、ユーザー・フックのパッケージ・パラメータから即時に入手できないものもあります。しかし、足りない値は人事管理システム表から導出することができます。
3. Professional ユーザー・インタフェースでフレックスフィールド検証が続行されるためには、フレックスフィールド・ポップアップ・ウィンドウが表示される前に、プロファイル値が設定されている必要があります。CUSTOM ライブラリを使用してこれを行うことができます。特定のイベントが発生した場合の特定のフォームについては、フォーム項目がカスタム・プロファイルに読み込まれます。

---

**注意：** Self Service 画面の中には、これらのフレックスフィールド値を表示できないものもある場合があります。これは、カスタム・プロファイルに値が正しく設定されるために必要な、フォームの CUSTOM ライブラリに相当する Web ページが存在しないためです。この問題は将来のリリースまでは解決されません。

---

### その他の解決策

値セットの WHERE 句を拡張して、データベースから必要な値を入手する方法もあります。このためには、他のデータベース表への結合が必要になることがあります。これにより、フォーム・ブロック・フィールド項目を参照する必要はなくなります。しかし、この方法は、データベース中にすでに存在しているレコードから値を入手できる場合にしか使用できません。現在の API コールによって処理中のレコード上の列を参照しようとすると失敗します。フレックスフィールド検証の実行時は、挿入操作中にデータベース表からこれらの値を使用することはできません。更新操作中には、更新前の値を入手できます。

## FND_SESSIONS 行の参照

FND_SESSIONS データベース表は、現在のユーザーのデートトラック有効日を入手するために使用されます。この表は、Professional ユーザー・インタフェースが単独で維持しているものです。API および Self Service モジュールは、この表で行の挿入も更新も行いません。このため、これらのモジュールから値セットが実行された場合、結合による行の検出は失敗します。

### 推奨する解決策

このデータベース・セッション用に FND_SESSIONS 表の中に行がまだ存在していない場合には、API「処理前」ユーザー・フックを使用して、行を挿入します。EFFECTIVE_DATE 列は、ユーザー・フックで使用できる p_effective_date パラメータから設定する必要があります。EFFECTIVE_DATE 列は、必ず時間コンポーネントを含まない日付値、つまり trunc(<date>) に設定することが重要です。そうしないと、結合条件によっては有効な表の行を検出できないことがあります。

API「処理前」ユーザー・フックによって FND_SESSIONS に行が挿入された場合、「処理後」ユーザー・フックでこれを削除する必要があります。これにより、同じ API を次にコールしたときに、FND_SESSIONS.EFFECTIVE_DATE 列が必ず正しい値に設定されているようにすることができます。

データのバッチ・アップロードのパフォーマンスが重要である場合には、最初の API コールの前に、バッチ・アップロード・プログラムで FND_SESSIONS 行を挿入した方が効率的な場合があります。この方法は、行のセットを同じ有効日で処理する場合にのみ使用できます。その場合でも、他のプログラムやインタフェースが必要な処理を行うためには、API ユーザー・フックの定義が必要になります。

### その他の解決策

フォーム・ブロック・フィールド項目の参照の場合と同じ、代替の解決策があります。FND_SESSIONS 表を使用して値セットで有効日を入手するかわりに、カスタム・プロファイルを使用できます。これにより、DML の挿入および削除のステップが省略されます。しかし、Professional ユーザー・インタフェースに対する影響があるため、CUSTOM ライブラリを変更してプロファイル値を設定する必要があります。

## 不完全なコンテキスト・フィールド値リスト

API を使用すると、フレックスフィールドの参照値がフレックスフィールドのコンテキスト・フィールド値リストに含まれていない場合、次のエラーが発生することがあります。

ORA-20001:CONTEXT とも呼ばれる列 ATTRIBUTE_CATEGORY には X という値は設定できません。

フレックスフィールドで business_group_id を参照フィールドとして使用したとします。API がコールされた場合、p_attribute_category パラメータは business_group_id の値に設定される必要があります。API はフレックスフィールドのコンテキスト・フィールド（ATTRIBUTE_CATEGORY）を検証するときに、使用されている business_group_id がフレックスフィールドのコンテキスト・フィールド値リストに存在しているかどうかを確認します。存在していない場合は、API からエラーが発行されます。

### 推奨する解決策

フレックスフィールドのコンテキスト・フィールド値リストに、考えられるすべての値が含まれていることを確認します。

### その他の解決策

フレックスフィールド体系によっては、グローバル・データ・エレメントのみが適用されるコンテキストがあることがあります（コンテキスト特有のセグメントはない）。これらのコンテキスト値を NULL 化するために、p_attribute_category パラメータを設定することも考えられます。これにより、コンテキスト・フィールド値リストにこれらのコンテキスト値をリストする必要はなくなります。しかし、他のデータ・エラーを検出できなくなる可能性があるため、この方法はお薦めできません。たとえば、特定の値を使用する必要のあるコンテキスト・フィールドが NULL に設定されていた場合は、その特定の値に関連する必須のセグメント検証は実行されません。

# 11

# オラクル人事管理システムにおけるセキュリティの拡張

## オラクル人事管理システムにおけるセキュリティの拡張

Oracle Human Resources では、表、レコード、フィールド、フォーム、機能へのアクセスを自由に制御できます。各従業員のアクセス・レベルを職責と一致させることも可能です。

オラクル人事管理システムのセキュリティについて、そしてユーザーの要件に応じたセキュリティの設定方法については、『Oracle HRMS ユーザーズ・ガイド』のセキュリティに関する章、およびこのガイドの「設定ステップ」を参照してください。

この章で、設定ステップやセキュリティの章の説明や定義を繰り返すことはしません。ここではこれらの情報に基づいて、セキュリティ・システムをインプリメントするオブジェクトおよびプロセスについて説明します。次の作業を行う場合には必ずお読みください。

- カスタム表を標準セキュリティ・システムに追加する
- 独自のセキュリティ・システムとシステム定義のメカニズムを統合する

## セキュリティ・プロファイル

Oracle Applications のユーザーはすべて、セキュリティ・グループおよびセキュリティ・プロファイルにリンクされた職責に基づいてシステムにアクセスします。セキュリティ・グループは、ユーザーがアクセスできるビジネス・グループを決定します。セキュリティ・プロファイルは、ビジネス・グループ内の、ユーザーがアクセスできるレコードを決定します（組織、職階、給与に関係する）。

セキュリティ・プロファイルには次の 2 種類があります。

- 無制限
- 制限付き

制限付きセキュリティ・プロファイルは、Oracle Human Resources、Oracle Payroll および Oracle Advanced Benefits のユーザーにのみ使用が認められています。Oracle Training Administration では制限付きセキュリティ・プロファイルは使用できないので注意してください。

無制限セキュリティ・プロファイルを持つ職責は、オラクル人事管理システムの表内のデータに無制限にアクセスできます。こうした職責は、APPS Oracle ユーザーに連結されています。無制限セキュリティ・プロファイルに接続した場合には、セキュア・ビューから選択した場合に表示されるデータと、そのセキュア・ビューの基盤となる表から選択した場合に表示されるデータは同じになります。

制限付きセキュリティ・プロファイルを使用して APPS Oracle ユーザーに接続した場合に、ご自分のセキュリティ・プロファイルに定義されているセキュリティ制限をバイパスしたければ、セキュリティ保護された表に直接アクセスすることもできます。必要に応じてそうすることにより、一意性チェックを実行したり、外部キーを解決することも可能です。

制限付きセキュリティ・プロファイルには、オプションで、読込み専用ユーザー、すなわちレポーティング・ユーザーを設定することも可能です。これらは、制限付きセキュリティ・プロファイルに 1 対 1 で関連付けられた個別の Oracle ユーザーで、Oracle 表やビューに対する読込み専用のアクセス権限を持ちます。レポーティング・ユーザーには、オラクル人事管理システム PL/SQL パッケージの実行権限はなく、セキュリティ保護されたオラクル人事管理システム表に直接アクセスすることもできません。

制限付きセキュリティ・プロファイルは、次のエンティティへのアクセスを制限することもできます（正確な制限は、セキュリティ・プロファイルの定義によって決まります）。

- 組織
- 個人情報
- アサイメント
- 職階
- 必要要員
- 給与

他のエンティティはすべて無制限です。つまり、制限付きセキュリティ・プロファイルは、前述のエンティティに関連する表、ビューおよび順序のあらゆるレコードにアクセスできます。

## セキュリティ保護された表およびビュー

次のオラクル人事管理システムの表はセキュリティ保護されています。

- HR_ALL_ORGANIZATION_UNITS
- PER_ALL_POSITIONS
- HR_ALL_POSITIONS_F
- PER_ALL_VACANCIES
- PER_ALL_PEOPLE_F
- PER_ALL_ASSIGNMENTS_F
- PAY_ALL_PAYROLLS_F

これらの表のうちの一部（すなわち、PER_ALL_PEOPLE_F、PER_ALL_ASSIGNMENTS_F、HR_ALL_POSITIONS_F および PAY_ALL_PAYROLLS_F）は、デートトラックされます。次ページの表は、上にリストされているセキュリティ保護された表を基盤とするビューを列挙したものです。

**表 11-1 セキュリティ保護された表およびビュー**

| 表またはビュー | 説明 |
|---|---|
| HR_ORGANIZATION_UNITS | 組織表のセキュリティ保護されたビュー |
| HR_ALL_ORGANIZATION_UNITS | 組織表 |
| PER_ORGANIZATION_UNITS | 組織ビューのセキュリティ保護されたビュー（人事システム組織のみ） |
| PER_ALL_ORGANIZATION_UNITS | 組織ビューのセキュリティ保護されていないビュー（人事システム組織のみ） |
| HR_ALL_POSITIONS | デートトラックされる職階表の無制限ビュー、セッション日に有効 |
| HR_ALL_POSITIONS_F | デートトラックされる職階表 |
| HR_POSITIONS | デートトラックされる職階表のセキュリティ保護されたビュー、セッション日に有効 |
| HR_POSITIONS_F | デートトラックされる職階表のセキュリティ保護されたビュー |
| HR_POSITIONS_X | デートトラックされる職階表のセキュリティ保護されたビュー、システム日に有効 |
| PER_POSITIONS | デートトラックされない職階表のセキュリティ保護されたビュー |
| PER_ALL_POSITIONS | デートトラックされない職階表 |
| PER_VACANCIES | 必要要員表のセキュリティ保護されたビュー |
| PER_ALL_VACANCIES | 必要要員表 |

**表 11-1 セキュリティ保護された表およびビュー（続き）**

| 表またはビュー | 説明 |
|---|---|
| PER_ASSIGNMENTS | アサイメント表のセキュリティ保護されたビュー、セッション日に有効 |
| PER_ASSIGNMENTS_F | アサイメント表のセキュリティ保護されたビュー |
| PER_ASSIGNMENTS_X | アサイメント表のセキュリティ保護されたビュー、システム日に有効 |
| PER_ALL_ASSIGNMENTS | アサイメント表のセキュリティ保護されていないビュー、セッション日に有効 |
| PER_ALL_ASSIGNMENTS_F | アサイメント表 |
| PER_PEOPLE | 個人情報表のセキュリティ保護されたビュー、セッション日に有効 |
| PER_PEOPLE_F | 個人情報表のセキュリティ保護されたビュー |
| PER_PEOPLE_X | 個人情報表のセキュリティ保護されたビュー、システム日に有効 |
| PER_ALL_PEOPLE | 個人情報表のセキュリティ保護されていないビュー、セッション日に有効 |
| PER_ALL_PEOPLE_F | 個人情報表 |
| PAY_PAYROLLS | 給与表のセキュリティ保護されたビュー、セッション日に有効 |
| PAY_PAYROLLS_F | 給与表のセキュリティ保護されたビュー |
| PAY_PAYROLLS_X | 給与表のセキュリティ保護されたビュー、システム日に有効 |
| PAY_ALL_PAYROLLS | 給与表のセキュリティ保護されていないビュー、セッション日に有効 |
| PAY_ALL_PAYROLLS_F | 給与表 |

## 制限付きセキュリティ・プロファイルによるオラクル人事管理システム・データへのアクセス

APPS Oracle ユーザーに接続すれば、追加設定を一切行わずに、あらゆるオラクル人事管理システム・データベース・オブジェクトにアクセスできます。

これは、レポーティング・ユーザーには当てはまりません。レポーティング・ユーザーにオラクル人事管理システムの表やビューへのアクセスを付与するためには、次の 2 つの条件を満たしている必要があります。

- 表やビューごとに、パブリック・シノニムが存在している必要があります。パブリック・シノニムは、指し示す先の表およびビューと同じ名称になります。パブリック・シノニムはオラクル人事管理システムのインストール時に作成されます。
- レポーティング・ユーザーには、SECGEN 処理によって表やビューへのアクセス許可を付与しておく必要があります。レポーティング・ユーザーには、SELECT 許可のみが付与されます。SECGEN についての詳細は、次を参照してください。

## セキュリティ保護されたビューの機能

セキュリティ保護されたビューを通してみることのできる情報は、そのビューにアクセスするのに使用したセキュリティ・プロファイルの定義によって異なります。

制限付きセキュリティ・プロファイルを使用して接続した場合には、表示される情報は、組織、職階、個人情報および給与の非正規化リストから引き出されます。

これらのリストは、必要な場合にのみ使用されます。たとえば、給与リストは、全給与計算を表示することのできるセキュリティ・プロファイルに関しては空になります。また、限定された一連の従業員を除く全応募者を表示することのできるセキュリティ・プロファイルに関しては、該当者リストに従業員は含まれますが、応募者は含まれません。

「HR: 複数ビジネス・グループ間」プロファイル・オプションが 'N' の場合、セキュリティ保護されたビューは現在のビジネス・グループのみのデータを返します。

「HR: 複数ビジネス・グループ間」プロファイル・オプションが 'Y' の場合、セキュリティ保護されたビューはすべてのビジネス・グループのデータを返します。ただし、現在のセキュリティ・プロファイルによって適用される追加の制限に従います。

HR_ORGANIZATION_UNITS のセキュリティ保護されたビューのテキストを次に示します。

```
SELECT HAO.ORGANIZATION_ID, HAOTL.NAME .....
FROM HR_ALL_ORGANIZATION_UNITS HAO,
     HR_ALL_ORGANIZATION_UNITS_TL HAOTL
WHERE DECODE(HR_SECURITY.VIEW_ALL, 'Y', 'TRUE',
      HR_SECURITY.SHOW_RECORD
      ('HR_ALL_ORGANIZATION_UNITS',HAOTL.ORGANIZATION_ID))='TRUE'
    AND DECODE(HR_GENERAL.GET_XBG_PROFILE,
      'Y', HAO.BUSINESS_GROUP_ID,
      HR_GENERAL.GET_BUSINESS_GROUP_ID_ = HAO.BUSINESS_GROUP_ID
    AND HAO.ORGANIZATION_ID = HAOTL.ORGANIZATION_ID
    AND HAOTL.LANGUAGE = USERENV('LANG')
```

人事システムのセキュリティ論理はほとんどカプセル化され、PL/SQL パッケージ HR_SECURITY に入れられています。

HR_SECURITY.VIEW_ALL は、現在のセキュリティ・プロファイルの VIEW_ALL_FLAG の値を返します。

HR_SECURITY.SHOW_RECORD は、現在のセキュリティ・プロファイルが制限付きセキュリティ・プロファイルの場合にコールされます。これは、現在のセキュリティ・プロファイルが問題の行を表示できるものかどうかを検証します。

HR_GENERAL.GET_XBG_PROFILE は、「HR: 複数ビジネス・グループ間」プロファイル・オプションの値を返します。

HR_GENERAL.GET_BUSINESS_GROUP_ID は、現在のビジネス・グループ ID を返します。

## セキュリティ・コンテキスト

人事システムのセキュリティ・コンテキストには、現在のセキュリティ・プロファイルのあらゆる属性の値が含まれています。このコンテキストは、PL/SQL グローバルを使用してインプリメントされます。現在のセキュリティ・プロファイルは次のようにして導出されます。

1. Oracle Applications サインオン画面を使用して Oracle Applications にログオンした場合には、Oracle Applications サインオン手順の一環としてセキュリティ・コンテキストが自動的に設定されます。ユーザーの現在の security_profile_id は、サインオン時に選択された職責とセキュリティ・グループから導出されます。
2. 人事システムのレポーティング・ユーザーに接続した場合には、現在のセキュリティ・プロファイル ID は、PER_SECURITY_PROFILES 表から導出されます。この表では、REPORTING_ORACLE_USERNAME が接続した Oracle ユーザーの名前と同じになります。

3. 前述の 2 つの方法のいずれでも security_profile_id を導出できない場合には、システムはビジネス・グループ用に作成されたデフォルトのビュー、全表示セキュリティ・プロファイルを参照します。これにより、ビジネス・グループに対する無制限アクセスが可能になります。これが見つからない場合には、現在の security_profile_id は NULL に設定され、ユーザーはどのレコードにもアクセスできなくなります。

このため、SQL*Plus によって APPS Oracle ユーザーに直接接続した場合には、人事管理システムの表に無制限にアクセスできます。しかし、人事システムのレポーティング・ユーザーに接続した場合には、ご自分のセキュリティ・プロファイルの定義に応じてアクセスが制限されます。

シミュレートするサインオン・セッションのユーザー、職責、アプリケーションおよびセキュリティ・グループの ID を渡して FND_GLOBAL.APPS_INITIALIZE（user_id、resp_id、resp_appl_id および security_group_id）をコールすることによって、Oracle Applications セッションのセキュリティ・コンテキストをシミュレートできます。security_group_id のデフォルトは 0（ゼロ）です（つまりビジネス・グループの設定）。

---

**注意：** 人事システムのレポーティング・ユーザーは FND_GLOBAL にはアクセスできません。

---

## セキュリティ・リスト

セキュリティ・プロファイル・リスト表には、個人情報、職階、組織および給与の非正規化リストが含まれています。従業員の退職時や応募の終了時には、追加セキュリティ・プロファイル・リスト表（PER_PERSON_LIST_CHANGES）に挿入されるので、退職従業員や前応募者の参照が引き続き可能です。PERSON_LIST 表は、現在の従業員と応募者のみを参照します。

セキュリティ・プロファイル・リストは、セキュリティ・プロファイルとセキュリティ保護された表の間の交差部表です。次を参照してください。

| セキュリティ・リスト表名 | 列 |
|---|---|
| PER_PERSON_LIST | SECURITY_PROFILE_ID, PERSON_ID |
| PER_POSITION_LIST | SECURITY_PROFILE_ID, POSITION_ID |
| PER_ORGANIZATION_LIST | SECURITY_PROFILE_ID, ORGANIZATION_ID |
| PAY_PAYROLL_LIST | SECURITY_PROFILE_ID, PAYROLL_ID |
| PER_PERSON_LIST_CHANGES | SECURITY_PROFILE_ID, PERSON_ID |

これらの表は、LISTGEN 処理によって定期的にリフレッシュされます。また、従業員、従業員の採用または異動について、Oracle Human Resources によって関連するビジネス・プロセスが実行された場合には、これらの表に書込みが行われます。

管理者階層および組織を通じて個人がセキュリティで保護されている場合は、職階と給与はセキュリティで保護されず、前述したセキュリティ・リスト表は使用されず、LISTGEN を実行する必要はありません。表示される個人のリストは、現在のユーザーに基づいて動的に導出されます。

ただし、組織、職階または給与（あるいはいずれかの組合せ）のセキュリティと組み合せて管理者セキュリティが使用されている場合は、定期的に LISTGEN を実行して、セキュリティ・リスト表をリフレッシュする必要があります。表示される個人のリストは、現在のユーザーに基づいて動的に導出され、PER_PERSON_LIST 表を通じて表示される個人のサブセットになります。

## セキュリティ処理

オラクル人事管理システムにセキュリティをインプリメントするには、次の 3 つの処理を使用します。

- セキュア・ロールに対する権限付与（ROLEGEN）
- セキュア・ユーザーの生成（SECGEN）
- セキュリティ・リストの作成（LISTGEN）

ROLEGEN は、インストールまたはアップグレードの一環として自動的に実行されます。レポーティング・ユーザーを設定しない場合は、SECGEN を実行する必要はありません。

「要求の発行」ウィンドウから SECGEN および LISTGEN を発行する方法の詳細は、『Oracle HRMS ユーザーズ・ガイド』のセキュリティに関する章を参照してください。本項では、これらの処理がどのような働きをするかについて説明します。

---

**注意：** GLISTGEN と呼ばれるセキュリティ・プロセスもあります。このプロセスは、グローバル・セキュリティ・プロファイルのリストを生成するために使用されます。これらは、ビジネス・グループとは関連付けられないセキュリティ・プロファイルです。これは、グローバルな（ビジネス・グループをまたがる）組織階層によって組織および個人を保護するものです。

---

### ROLEGEN: セキュア・ロールに対する権限付与処理

ロールとは、Oracle ユーザーまたはその他のロールに対して付与することのできる一連の権限のことをいいます。ロールは、SECGEN 処理によってユーザーに付与されます（次項を参照）。

ROLEGEN プロセスは SECGEN を実行する前に実行する必要があります。ROLEGEN は、HR_REPORTING_USER というロールに対して、オラクル人事管理システムの表やビューに関する SELECT 権限を動的に付与します。ROLEGEN を実行する前にこのロールが存在している必要があります。

HR_REPORTING_USER ロールはオラクル人事管理システムのインストール中に作成されます。さらに、ROLEGEN もオラクル人事管理システムのインストール中に実行されます。

> **注意：** ROLEGEN はインストールおよびアップグレード処理の一環として実行されるため、ROLEGEN を手動で実行する必要はありません。

ROLEGEN は、次の処理を実行します。

- 無保護の表を除いて、人事管理システムの表およびビューのパブリック・シノニムを作成する（%_ALL_%）
- HR_REPORTING_USER ロールから既存の権限をすべて取り消す
- 人事管理システムの表やビューに関する SELECT 権限を HR_REPORTING_USER というロールに付与する

## SECGEN - セキュア・ユーザーの生成処理

指定されたセキュリティ・プロファイルに対して SECGEN を実行します。この処理では、セキュリティ・プロファイルに対応付けられた Oracle ユーザーに、HR_REPORTING_USER ロールの権限が付与されます。

SECGEN は、必ず ROLEGEN の後に実行してください。しかし、いったん特定のセキュリティ・プロファイルに対して SECGEN を実行してあれば、その後に ROLEGEN を再実行した場合でも、SECGEN を再実行する必要はありません。

SECGEN は組込みの SQL 文を含む PRO*C プロセスです。SECGEN は「要求の発行」ウィンドウから開始します。

## LISTGEN - セキュリティ・リストの作成処理

LISTGEN は、セキュリティ保護されたビューが作成される際の基盤となるセキュリティ・リストをリフレッシュするために、定期的に（たとえば毎晩）実行する必要があります

LISTGEN は、「要求の発行」ウィンドウから発行する PL/SQL プロシージャです。

LISTGEN は、PER_ORG_STRUCTURE_ELEMENTS や PER_POS_STRUCTURE_ELEMENTS といった表のツリー走査を行うことによって、組織階層や職階階層からセキュリティ・リストを作成します。ノード間の親子関係を用いて、指定されたトップ・ノードから始められます。有効実行日としてこの処理に渡された日付時点のバージョンの階層が使用されます。

LISTGEN は、セキュリティ・プロファイルごとに、最高組織として指定されている組織が現バージョンの階層に存在しているかチェックします。存在していない場合には、LISTGEN はエラー・メッセージをログ・ファイルに書き込み、エラー状態で失敗に終わります。このような事態は、セキュリティ・プロファイルの最高組織として参照される組織が新バージョンの階層に含まれていなかった場合に発生する可能性があります。

指定された場合には、最高職階についても同様のチェックが行われます。

各セキュリティ・プロファイルに対して、LISTGEN は次のステップを実行します。

1. 「全表示」フラグが Y の場合には、LISTGEN は指定されたセキュリティ・プロファイルのすべてのセキュリティ・リストを空のままの状態にしておくことを止めます。

2. 給与計算リストを作成します。

   「全給与計算の表示」フラグが Y の場合は、LISTGEN は給与計算リストを空のままの状態にしておきます。「全給与計算の表示」フラグが N の場合は、LISTGEN は「給与計算に含む」フラグをチェックします。このフラグが Y の場合は、LISTGEN は pay_security_payrolls リスト内に全給与計算のリストを作成します。このフラグが N の場合は、LISTGEN は pay_security_payrolls リストの給与計算を除くすべての給与計算のリストを作成します。「セキュリティ・プロファイルの定義」画面で給与計算を入力すると、pay_security_payrolls リストに挿入されます。

3. 組織リストを作成します。

   「全組織の表示」フラグが Y の場合は、LISTGEN は組織リストを空のままの状態にしておきます。このフラグが N の場合は、LISTGEN は、ユーザーが「セキュリティ・プロファイルの定義」画面で選択した組織階層に対して指定した最高組織より下の全組織のリストを作成します。「最高組織に組込む」フラグが Y の場合は、指定した最高組織がリストに含まれます。ビジネス・グループは必ずリストに含まれるため、組織に割り当てられる前に、新たに入力された従業員と応募者を参照することができます。

4. 職階リストを作成します。

   「全職階の表示」フラグが Y の場合は、LISTGEN は組織リストに掲載されている組織内の全職階のリストを作成します。このフラグが N の場合は、LISTGEN は、ユーザーが「セキュリティ・プロファイルの定義」画面で選択した職階階層に対して指定した最高職階より下の全職階のリストを作成します。「最高職階に組込む」フラグが Y の場合は、指定した最高職階がリストに含まれます。職階のリストは、組織リストのすべての組織に対して、または「全組織の表示」フラグが Y の場合には全組織に対して作成されます。

5. 該当者リストを作成します。

   「全職階の表示」フラグが N の場合は、LISTGEN は、職階リスト内の職階に対する現在のアサイメントを持つすべての従業員または応募者のリストを作成します。ただし、それらの従業員または応募者が給与計算リストから除外された給与にも割り当てられている場合は除きます。LISTGEN は、職階に割り当てられていないものの、給与計算リスト内の給与、もしくはいずれかの給与（「全給与計算の表示」フラグが Y の場合）に割り当てられている個人情報もリストに含めます。

   このリスト内の個人情報は、LISTGEN に渡された日付時点のアサイメントを持つか、前に終了したアサイメントを持たない将来着任する個人に対する最初のアサイメントです。このため、将来の新規着任者をセキュア・ビューを通して参照することができます。

「全組織の表示」フラグが N で、「全職階の表示」フラグが Y の場合は、LISTGEN は、組織リスト内の組織に対する現在のアサイメントを持つすべての個人情報のリストを作成します。「全給与計算の表示」フラグが N の場合は、リストは給与計算リスト内の給与に対するアサイメントを持つ個人情報、または給与アサイメントを持たない個人情報に限定されます。

どのセキュリティ・プロファイルに関しても、該当者リストにはまだ未割当ての個人情報が含まれます。

6. 該当者リストの変更を追加します。

   退職または応募終了時にセキュリティ・プロファイルが参照できる従業員または応募者は、退職または応募終了後も引き続き参照できることが求められます。そのため、「退職」または「応募終了」フォームは、退職または応募終了時にその個人情報を参照できる各セキュリティ・プロファイルの該当者リスト変更表に行を挿入します。

   PER_PERSON_LIST_CHANGES 表に入力があり、現在その個人に関して勤務期間ならびに応募がない場合には、LISTGEN はその個人を該当者リストに追加します。ただし、該当者リストに存在していない場合にのみ追加されます。

## カスタム表のセキュリティ保護

独自のカスタム表を作成した場合には、次のステップを実行して、レポーティング・ユーザーがそれらの表示にアクセスできるようにします。

1. 表を作成します。

   Oracle Applications に存在している可能性のあるいかなる表またはビューともかち合わない表名を選択してください。

   HR、PER、PAY、FF、DT、SSP、GHR、BEN、OTA、HXT、EDW、HRI、HXC、PQH、PQP または IRC などの 2 文字または 3 文字からなる接頭辞は使用しないでください。

2. カスタム表を所有するユーザーから、表に対する SELECT アクセスを HR_REPORTING_USER ロールに付与します。

   ```
   GRANT SELECT ON custom_table TO hr_reporting_user;
   ```

   インストールまたはアップグレードを行うたびに、このステップを繰り返す必要があります。ただし、HR_REPORTING_USER というロールに対してすでにアクセス権が付与されている既存のレポーティング・ユーザーはそのロールに追加された新規権限を自動的に受け取るので、SECGEN を再実行する必要はありません。

3. その表のシノニムを作成します。

   パブリック・シノニムを使用する場合は、ユーザーがパブリック・シノニムの作成を開始するOracleユーザーが「パブリック・シノニムの作成」というシステム権限を持っている必要があるということを覚えておいてください。

```
CREATE PUBLIC SYNONYM custom_table
  FOR base_table_account.custom_table;
```

# 12

# オラクル人事管理システムの API

## オラクル人事管理システムの API

アプリケーション・プログラム・インタフェース（API）は、外部プロセス・ルーチンを論理的にグループ化したものです。オラクル人事管理システムでは、一連の PL/SQL パッケージ・プロシージャおよび関数を通じてデータベースへのオープン・インタフェースが提供されています。便宜上、弊社ではこれらのパッケージになったプロシージャの 1 つ 1 つを API と呼んでいます。

ここでは、これらの API を使用するのに必要な技術的な情報をすべて提供するとともに、次の事柄について説明します。

- API の概要 : 12-3 ページ

  オラクル人事管理システムの API の使用方法およびこの方式の利点を説明します。

- オブジェクト・バージョン番号（OVN）について : 12-5 ページ

  オブジェクト・バージョン番号の役割を説明します。API はこの番号を使用して、別のユーザーが行を更新していないかを確認することにより、行の変更の上書きを防止します。

- API パラメータ : 12-7 ページ

  各 API で使用されるパラメータに関する情報の検索方法、パラメータの命名規則、API コールにおいてパラメータ・リストの順序に頼るかわりにパラメータに名前を付けることの重要性、そしてすべてのパラメータを指定する必要のないようにデフォルト値を使用する方法を説明します。また、デートトラック操作を制御するパラメータなど、特定の制御パラメータの操作についても説明します。

- API 機能 : 12-22 ページ

  コミットは、API ではなくコール側プログラムによって処理されることを説明し、この方法の利点を説明します。また、同じコミット単位内で複数の API をコールしたときにデッドロックを回避する方法も説明します。

- API のフレックスフィールド : 12-23 ページ

  キー・フレックスフィールドおよび付加フレックスフィールドの値が API によってどのように検証されるかを説明します。

- 複数言語サポート : 12-24 ページ

  複数言語サポート API の使用方法を説明します。

- 代替 API : 12-25 ページ

  住所の作成など、特定のビジネス・プロセス用に、国別仕様の API が提供されていることを説明します。

- API エラーと警告 : 12-26 ページ

  API がエラーおよび警告を発する方法と、コール側コードがそれらを処理する方法を説明します。バッチ処理のエラーを処理するためにメッセージ表が提供されています。

- PL/SQL バッチ・プログラム例 : 12-28 ページ

  個人の住所データのバッチ・ロードの方法と、検証エラーの処理方法を説明します。

- WHO 列と Oracle Alert : 12-31 ページ

  API を使用して WHO 列（データベースの行の作成または更新を行ったアプリケーション・ユーザーを記録する）を作成する方法を説明します。

- API ユーザー・フック : 12-32 ページ

  ユーザー・フックは、API に処理ロジックまたは検証を追加するための場所です。API 内には、特定のビジネス・プロセスに関連する検証を追加するためのフックがあります。また、特定のデータ項目の検証のために、表レベルのモジュール内にもフックがあります。この項では、ユーザー・フックを使用できる場所と、そのインプリメント方法を説明します。また、データベース・トリガーと比較した利点についても説明します。

- 基本ブロックとしての API の使用方法 : 12-54 ページ

  システムから提供されている API をコールするユーザー独自の API の作成方法を説明します。

- Oracle Forms におけるオブジェクト・バージョン番号の処理方法 : 12-55 ページ

  API をコールするユーザー独自のフォームを作成する場合に、オブジェクト・バージョン番号を管理する追加のフォーム・ロジックをインプリメントする方法を説明します。

# API の概要

オラクル人事管理システムの API を設計する場合には、あらゆるデータ処理作業を簡略化すると同時に、アップグレード時の顧客の拡張を保護する隔離層をユーザーとデータ・モデルの間に提供する API を設計するということが基本となります。API は、完全なデータの妥当性検査と処理を提供するパラメータ付きの実行可能な PL/SQL パッケージです。

API レイヤーを使用することにより、ユーザー・インタフェース・レイヤーのみではなく、データベース内のビジネス・ルールの取込みおよび実行が可能になります。このレイヤーは、Web ページやスプレッドシートなどの人事管理システムへの代替インタフェースの使用をサポートするとともに、すべてのトランザクションがシステムにインプリメントされているビジネス・ルールに準拠していることを保証します。このレイヤーはまた、オラクル人事管理システムと他のシステムまたはプロセスとの統合を簡略化するだけでなく、初期ロードもサポートします。

## 代替ユーザー・インタフェース

サポートされている API は、オラクル人事管理システムへの代替データ・エントリ・ポイントとして使用できます。オンライン・フォームを使用して手動で新規情報を入力したり既存データを変更するかわりに、同じ操作を実行する別のプログラムをインプリメントできます。

これらの別プログラムは、データベース内のデータを直接変更することはありません。これらのプログラムは次のような機能の API をコールします。

1. 特定の業務処理を許可することが適切であることを保証する
2. API に渡されたデータの妥当性を検査する
3. 人事システム・スキーマのデータを挿入 / 更新 / 削除する

API は、サーバー側にインプリメントして、いろいろな使用方法ができます。例 :

- 既存のシステムからデータをアップロードする顧客。データのタイプ入力をする臨時データ入力事務員を雇うかわりに、既存のシステムからデータを抽出した後で、API をコールして抽出したデータをオラクル人事管理システムに転送するプログラムを記述することができます。
- 様々なベンダーから多数のアプリケーションを購入して完全なソリューションを構築する顧客。統合された環境においては、1 つのアプリケーションに変更を加えると、別のアプリケーションのデータにも変更を加える必要がある場合もあります。ユーザーが絶え間なく変化するアプリケーションを忘れずに 1 つ 1 つ精査するかわりに、人事管理システム・アプリケーションを電子的に更新することができます。変更はバッチ単位で、もしくは個別に即座に行うことができます。
- オラクル人事管理システムのシステム定義の標準フォームのカスタム・バージョンを作成する顧客。API を使用して 1 つ以上のフォームの代替バージョンをインプリメントすることにより、すべてのデータベース・トランザクションを管理することができます。

- Web ベースのインタフェースを開発して、標準オラクル人事管理システム・フォームの配置またはサポートに費用を掛けずに、一時ユーザーが人事システム情報にアクセスし、保持できるようにすることを望む顧客。これは、従業員による氏名、住所、婚姻区分の変更などの本人情報の問合せや更新を可能にする Self-Service 機能の大半の基礎となっています。これはまた、管理する従業員の詳細を問合せまたは保持するマネージャにも適用されます。
- スプレッドシート・アプリケーションの使用に慣れているマネージャは、必要に応じて、データベースにさえも接続せずにデータをエクスポートおよび処理し、再接続時に人事管理システム・データベースに変更内容をアップロードすることもできます。

前述のいずれの例の場合も、プログラムによってオラクル人事管理システム・データベースの表のデータを直接変更する必要はないでしょう。特定のプログラムが 1 つ以上の API をコールし、コールされた API がオラクル人事管理システム・データベースに無効なデータが書き込まれたり、既存のデータが壊されることがないように保護します。

## API を使用することの諸利点

なぜデータベースの表のデータを直接変更せずに API を使用するのでしょうか？

オラクル社は、PL/SQL を使用したアプリケーション内のデータの直接操作は一切サポートしていません。API を用いた場合の諸利点を次に示します。

- API を使用することにより、Oracle Forms を使用せずに人事システムおよび給与システム情報を保持できます。
- API を用いれば、データベース構造のすべての機能を完全に理解する必要はありません。API が表間の関連やアップデートをすべて管理します。
- API は、データベースの整合性を保持することが保証されています。必要であれば、データベースの行レベル・ロックを使用して、各種表の間の一貫性を確保します。無効なデータをシステムに入力することができないため、既存のデータが誤った変更から保護されます。
- API は、ビジネス・プロセスのすべての部分をデータベースに適用することが保証されています。API がコールされると、そのトランザクション全体が完了するか、個々のデータベースの変更がすべて適用されます。トランザクションが完了しなかった場合には、API がコールされなかったかのように、データベースは最初の有効な状態のままになります。
- API は、コミットを発行することにより、こうした変更を永続的なものにしません。変更を永続的なものにするのは、コール側プログラムです。このため、個別レコードの処理とバッチ処理を自由に使用できます。さらに、Forms などのクライアント・プログラムによって実行される標準コミット処理に影響が及ばないようにします。

- API を用いれば、顧客固有の論理をアップグレード時のデータベース構造の変更から簡単に守ることができます。オラクル社は、機能の向上や拡張をサポートするために API に変更を加えることは一切ないとは保証できませんが、変更の数を最小限に抑えること、そうした変更が発生した場合には、適切な通知や文書を提供することをコミットします。

> **注意：** オラクル人事管理システムの API をコールするプログラムを書く場合は、PL/SQL バージョン 2 に関する知識が必要です。この項の残りの部分では、PL/SQL でのプログラミングに関する知識があることを前提に、API のコール方法について説明します。

## オブジェクト・バージョン番号（OVN）について

すべてのデータベースの表のほとんどすべての行に、オブジェクト・バージョン番号（object_version_number）が割り当てられています。新しい行が挿入されると、API は通常、オブジェクト・バージョン番号を 1 に設定します。データベース内のその行が更新される度に、オブジェクト・バージョン番号が増分されます。その行は、次に更新されるまで、もしくは削除されるまで、そのオブジェクト・バージョン番号を持ち続けます。この番号が減分されたり、前の値にリセットされることはありません。

> **注意：** オブジェクト・バージョン番号は一意ではないため、主キーを置き換えるものではありません。1 つの表に同じバージョン番号を持つ行を多数含めることができます。オブジェクト・バージョン番号は、特定の主キーの行のバージョンを表します。

データベース行がクライアントに転送（問合せ）されるときには必ず、既存のオブジェクト・バージョン番号が他の属性と一緒に転送されます。オブジェクトがクライアントによって変更され、サーバーに保存し直された場合には、現在のサーバー・オブジェクト・バージョン番号がクライアントから渡された値と比較されます。

- 2 つのオブジェクト・バージョン番号の値が同じ場合、サーバー側の行は、属性がクライアントに転送されたときと同じ状態にあります。他には一切変更がなかったので、現在の変更要求は続行可能です。このため、オブジェクト・バージョン番号は増分されます。
- 2 つの値が異なる場合には、別のユーザーがサーバー側の行をすでに変更し、コミットしています。他のユーザーが行った変更が上書きされて失われる可能性があるため、現在の変更要求の続行は許されません。（データベース・ロックは、コミット前の変更が他のユーザーにより上書きされることを防止するために使用されます。）

オブジェクト・バージョン番号を使用して、オンライン・システムと同様の妥当性検査比較を行うことができます。Forms は、あらゆるフィールド値を対話形式で比較し、なんらかの差異が検出された場合には、「レコードは別のユーザーによって変更されました」というエラー・メッセージを表示します。オブジェクト・バージョン番号を用いれば、データベースを長期にわたってロックし続けることなく、長期間にわたってトランザクションを発生させることができます。たとえば、クライアント・アプリケーションは、行を局所的に保存し、サーバーから切断して、後日に再度接続して変更内容をデータベースに保存することも可能です。その上、ユーザーがクライアントやサーバーの値をすべてチェックする必要もありません。

**例**

個人の新規住所を作成するとしましょう。「*create_person_address*」API は、新規データベース行の object_version_number（オブジェクト・バージョン番号）を 1 に自動設定します。その後で、2 人のユーザーが同時にこの住所を問合せます。ユーザー A とユーザー B には、現在の object_version_number が 1 の同じ住所詳細が表示されます。

ユーザー A は「町」フィールドを別の値に更新し、「*update_person_address*」API をコールして、現在の object_version_number 1 を渡します。この object_version_number はデータベース行の値と同じであるため、更新が許可され、object_version_number が 2 に増分されます。新しい object_version_number がユーザー A に返され、行がデータベースにコミットされます。

元の行の詳細を持っているユーザー B は、住所の第 1 行目が誤っていることに気が付きます。ユーザー B は「update_person_address」API をコールして、第 1 行目を新たに渡します。ユーザー B は、現在の object_version_number は（1）であると思っています。API は、この値をデータベース行の現在の設定値である（2）と比較します。差異があるため、更新の続行は許可されず、エラーがユーザー B に返されます。

問題を解決するため、ユーザー B はこの住所を再度問合せ、新しい町を確認し、object_version_number 2 を取得します。住所の第 1 行目が更新され、「update_person_address」API が再度コールされます。object_version_number がデータベース行の値と同じであるため、更新の続行が許可されます。

その結果、最初の変更が上書きされることなく、両方の更新が適用されます。

## API 制御パラメータ p_object_version_number について

公開済 API の大多数には p_object_version_number 制御パラメータがあります。

- スタイル作成 API の場合、このパラメータは OUT と定義され、常に初期化されます。
- スタイル更新 API の場合、このパラメータは IN OUT と定義されます。このパラメータは必須です。

この API は、オブジェクト・バージョン番号とデータベース内の現在の設定値が必ず一致するようにします。値が一致していない場合は、アプリケーション・エラー HR_7155_OBJECT_LOCKED が生成されます。API コールの終わりにエラーがなければ、新規オブジェクト・バージョン番号が渡されます。

オブジェクトがデートトラックされていないときのスタイル削除 API の場合、このパラメータは必須の IN パラメータとなります。また、オブジェクトがデートトラックされているときのスタイル削除 API の場合、これは必須の IN OUT パラメータとなります。

この API は、オブジェクト・バージョン番号とデータベース内の現在の設定値が必ず一致するようにします。値が一致していない場合は、アプリケーション・エラー HR_7155_OBJECT_LOCKED が生成されます。リストに残っているデートトラック・オブジェクトにエラーがなければ、新しいオブジェクト・バージョン番号が渡されます。

関連項目 :

p_datetrack_update_mode 制御パラメータについて : 12-18 ページ

p_datetrack_delete_mode 制御パラメータについて : 12-19 ページ

Oracle Forms におけるオブジェクト・バージョン番号の処理方法 : 12-55 ページ

### オブジェクト・コンフリクトの検出および処理

処理対象行に正しいオブジェクト・バージョン番号が設定されていない場合には、アプリケーション・エラー HR_7155_OBJECT_LOCKED が生成されます。このエラーは、ユーザーがその情報を選択したために、特定の行の変更およびコミットが完了したことを示します。他の変更内容が誤って上書きされないようにするためには、その情報を再度選択し、変更内容を再適用してから、API に再発行します。

## API パラメータ

この項では、オラクル人事管理システムにおけるパラメータの使用方法について説明します。

### パラメータ情報の探索

パッケージ・ヘッダー作成スクリプト内のドキュメンテーションを調べるか、SQL*Plus を使用することによって、各 API のパラメータを見つけ出すことができます。

#### パッケージ・ヘッダー作成スクリプト

IN パラメータと OUT パラメータの一覧を含む、各 API の説明については、パッケージ・ヘッダー作成スクリプト内のドキュメンテーションを参照してください。

メイン・リリースの最初のバージョンに含まれるコア製品 API の場合、スクリプトは製品の上位の admin/sql ディレクトリに入っています。*api.pkh などのファイル名を参照してください。ローカライズ独自の API には *LLi.pkh という命名標準が使用されます。この LL は 2 文字のローカライズ・コードです。

たとえば、hr_employee_api パッケージ内の全 API についての詳細は、$PER_TOP/admin/sql/peempapi.pkh ファイルを参照してください。

メイン・リリースの最初のバージョンに搭載されていなかった新しい API、または国別仕様固有の API については、オペレーティング・システムの違うディレクトリにある場合もあります。

Oracle では次のマニュアルに記載されている API のみサポートしています。

- 『Oracle HRMS ユーザーズ・ガイド』およびヘルプ・システムの「パブリック・コールが可能なビジネス・プロセス API」のトピック。
- ヘルプ・システムの「オラクル人事管理システムの新機能」のトピック。メイン・リリースの最初のバージョン後に採用された新規 API はここに記載されています。

これらのリストは、次の 3 つの基準すべてを満たす限定されたサーバー側コードです。

- データベース・パッケージ名が "_API" で終わる。
- パッケージ・ヘッダー作成スクリプトのファイル名は *api.pkh または *LLi.pkh という命名標準に従う。この LL は 2 文字の国別仕様コード。
- 個々の API ドキュメンテーションに、「パブリック」の値を持つ「アクセス」セクションがある。

他の多くのパッケージには、API コードそのものによってコールすることのできるプロシージャや関数が含まれています。特に明記されていないかぎり、他のルーチンの直接コールは一切サポートされていません。これは、API 検証および論理ステップがバイパスされるためです。このため、オラクル人事管理システム・アプリケーション・スイート内のデータが破損する可能性があります。

#### パラメータをリストするための SQL*Plus の使用方法

PL/SQL パラメータのリストのみが必要な場合は、SQL*Plus を使用します。SQL*Plus プロンプトで、describe コマンドを使用します。その後に、データベース・パッケージ名、期間、API 名の順に入力します。たとえば、「create_grade_rate_value」API のパラメータをリストするには、SQL の > プロンプトで次のように入力します。

```
describe hr_grade_api.create_grade_rate_value
```

## パラメータ名

各 API は、いくつかのパラメータを持ちます。これらのパラメータは指定される場合も、指定されない場合もあります。ほとんどのパラメータは、人事システム・スキーマのデータベースの列にマップされています。データベース上に明示的に保持されていない処理ロジックに影響を及ぼす制御パラメータもいくつかあります。

パラメータ名は必ず *p_* で始まります。パラメータがデータベースの列にマップされている場合には、名前の残りの部分は通常は列名と同じになります。長さが 30 文字に制限されているため、名前の中には切り捨てられるものもあります。パラメータとそれに対応するデータベースの列名の両方がコードの同一セクションで参照される場合のコーディング・コンフリクトを避けるために、*p_* を接頭辞として付けることによって、パラメータ名に実際の列名とわずかな違いを持たせています。

パラメータ間で命名コンフリクトが発生する場合は、データベース・エンティティを特定する 3 文字の短縮コードがパラメータ名に含められます。実際には名前に矛盾がない場合も時にはありますが、そのパラメータが関連付けられているエンティティについての混乱を避けるために、3 文字の短縮コードが使用されます。

たとえば、create_employee には、こうしたケースの両方の例が含まれています。新規従業員の作成ロジックの一部に、個人レコードの挿入とアサイメント・レコードの挿入があります。これらのエンティティはいずれも、object_version_number を持ちます。API は 2 つの OUT パラメータを使用して、両方の object_version_number 値を返します。両方のパラメータを p_object_version_number と呼ぶことはできないため、p_per_object_version_number が個人レコードの値を、p_asg_object_version_number がアサイメント・レコードの値をそれぞれ取ります。

どちらのエンティティにも、関連するテキスト・コメントを入れることができます。なんらかのコメントが「create_employee」API に渡された場合、それらのコメントは個人レコードに対して付けられます。アサイメント・レコードに対するコメントはブランクのままです。

データベース内のコメントが指定した場所についての混乱を避けるために、API は p_per_comment_id パラメータを使用して ID を返します。

## パラメータの名前表記法

API をコールするときには、「位置表記法」のかわりに「名前表記法」を使用することをお薦めします。つまり、パラメータのリスト順序に頼るのではなく、各パラメータの名前をコール内にリストします。

「名前表記法」の使用により、パラメータ・インタフェースの変更からコードを保護することができます。将来のリリースでは、「名前表記法」によって、パラメータの API への追加時や API からの削除時のコードの保守が容易になります。

たとえば、次のプロシージャ宣言について考えてみましょう。

```
procedure change_age
  (p_name     in     varchar2
  ,p_age      in     number
  ;
```

「名前表記法」によるコール:

```
begin
  change_age
    (p_name => 'Bloggs'
    ,p_age  => 21
    );
end;
```

「位置表記法」によるコール:

```
begin
  change_age
    ('Bloggs'
    ,21
    );
end;
```

## デフォルト・パラメータ値の使用方法

API をコールするときに必ずしもすべてのパラメータを指定する必要はありません。PL/SQL デフォルト値が指定されている場合は、値の指定はオプションです。

ユーザー定義のフォームから API をコールする場合には、API コール内にすべてのパラメータを指定する必要があります。フォームからのサーバー側 PL/SQL のコールにおいては PL/SQL 宣言デフォルト値はサポートされていないため、PL/SQL 宣言デフォルト値を使用することはできません。

### スタイル作成 API のデフォルト・パラメータ

人事システム・スキーマ内に新規データを作成する API の場合、オプション・パラメータは通常デフォルト値 NULL と見なされます。妥当性検査が完了すると、対応するデータベースの列が NULL に設定されます。API をコールするときは、デフォルト値が定義されていないパラメータをすべて指定する必要があります。

しかし、API の中には、いくつかの属性値を導出するための論理が含まれているものもあります。ユーザーが PL/SQL デフォルト値を渡すと、API はそのデータベースの列に設定する特定の値を決定します。それでも、NULL 値を渡したり、コール内にパラメータを指定するのを止めるかわりに、ユーザー定義の値を渡すことによって、この API ロジックを上書きすることも可能です。

IN OUT パラメータはコール側パラメータ・リストに必ず含める必要があるので、注意が必要です。API は値を渡すことができるため、この種のパラメータには変数を使用して値を渡す必要があります。

これらの変数には、API をコールする前にユーザー定義の値を指定してください。IN OUT パラメータの値を指定しない場合は、パラメータに NULL 値を渡すために変数を使用します。

---

**注意:** 各 IN OUT パラメータで NULL 値を設定できるかどうかは、各 API パッケージ・ヘッダー作成スクリプトのコメントで確認してください。

---

「create_employee」API には、これらの各種パラメータすべての例が含まれています。

```
procedure create_employee
  (
  ...
  ,p_sex                          in     varchar2
  ,p_person_type_id               in     number
                                    default null
  ...
  ,p_email_address                in     varchar2
                                    default null
  ,p_employee_number              in out varchar2
  ...
  ,p_person_id                       out number
  ,p_assignment_id                   out number
  ,p_per_object_version_number       out number
  ,p_asg_object_version_number       out number
  ,p_per_effective_start_date        out date
  ,p_per_effective_end_date          out date
  ,p_full_name                       out varchar2
  ,p_per_comment_id                  out number
  ,p_assignment_sequence             out number
  ,p_assignment_number               out varchar2
  ,p_name_combination_warning        out boolean
  ,p_assign_payroll_warning          out boolean
  ,p_orig_hire_warning               out boolean
  );
```

PL/SQL デフォルト値が 1 つも定義されていないため、p_sex パラメータを設定する必要があります。p_person_type_id パラメータには、従業員の Person タイプ ID を渡すことができます。値を指定しなかたった場合や、NULL 値を明示的に渡した場合には、API はこのデータベースの列を、そのビジネス・グループのアクティブ・デフォルト従業員システムの Person タイプ ID に設定します。API パッケージ・ヘッダー作成スクリプトのコメントはさらに詳しい情報を提供します。

p_email_address パラメータに値を渡す必要はありません。コール内にこのパラメータを指定しないと、対応するデータベースの列に NULL 値が入れられます（これは、フォームのユーザーが表示フィールドをブランクのままにしておくようなものです）。

p_employee_number パラメータは、コールごとに指定してください。従業員番号を設定しない場合は、コール側ロジックで使用する変数を NULL に設定する必要があります（p_employee_number パラメータの場合、従業員番号の生成方法が手動に設定されているときには、ビジネス・グループの値を指定する必要があります。値は生成方法が自動または国別識別子の場合にのみ渡されます）。

**例 1**

(1) ビジネス・グループの従業員番号生成方法が手動、(2) デフォルトの従業員 Person タイプが必要、(3)E メール属性の設定が不要である場合の「create_employee」API のコール例を次に示します。

```
declare
  l_emp_num                   varchar2(30);
  l_person_id                 number;
  l_assignment_id             number;
  l_per_object_version_number number;
  l_asg_object_version_number number;
  l_per_effective_start_date  date;
  l_per_effective_end_date    date;
  l_full_name                 varchar2(240);
  l_per_comment_id            number;
  l_assignment_sequence       number;
  l_assignment_number         varchar2(30);
  l_name_combination_warning  boolean;
  l_assign_payroll_warning    boolean;
  l_orig_hire_warning         boolean;
begin
  --
  -- Set variable with the employee number value,
  -- which is going to be passed into the API.
  --
  l_emp_num := 4532;
  --
  -- Put the new employee details in the database
  -- by calling the create_employee API
  --
  hr_employee.create_employee
    (p_hire_date                =>
                to_date('06-06-1996','DD-MM-YYYY')
    ,p_business_group_id        => 23
    ,p_last_name                => 'Bloggs'
    ,p_sex                      => 'M'
    ,p_employee_number          => l_emp_num
    ,p_person_id                => l_person_id
    ,p_assignment_id            => l_assignment_id
    ,p_per_object_version_number => l_per_object_version_number
    ,p_asg_object_version_number => l_asg_object_version_number
    ,p_per_effective_start_date  => l_per_effective_start_date
    ,p_per_effective_end_date    => l_per_effective_end_date
    ,p_full_name                => l_full_name
    ,p_per_comment_id           => l_per_comment_id
    ,p_assignment_sequence      => l_assignment_sequence
    ,p_assignment_number        => l_assignment_number
```

```
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    ,p_orig_hire_warning        => l_orig_hire_warning
    );
end;
```

**注意：** データベースの employee_number 列は、ビジネス・グループの employee_number 生成方法が国別識別子に設定されている場合を考慮して、varchar2 と定義されています。

### 例 2

(1) ビジネス・グループの従業員番号生成方法が自動、(2) 非デフォルトの従業員 Person タイプを使用する必要がある、(3)E メール属性詳細を保持する必要がある場合の「create_employee」API のコール例を次に示します。

```
declare
  l_emp_num                   varchar2(30);
  l_person_id                 number;
  l_assignment_id             number;
  l_per_object_version_number number;
  l_asg_object_version_number number;
  l_per_effective_start_date  date;
  l_per_effective_end_date    date;
  l_full_name                 varchar2(240);
  l_per_comment_id            number;
  l_assignment_sequence       number;
  l_assignment_number         varchar2(30);
  l_name_combination_warning  boolean;
  l_assign_payroll_warning    boolean;
  l_orig_hire_warning         boolean;
begin
  --
  -- Clear the employee number variable
  --
  l_emp_num := null;
  --
  -- Put the new employee details in the database
  -- by calling the create_employee API
  --
  hr_employee.create_employee
    (p_hire_date                =>
                 to_date('06-06-1996','DD-MM-YYYY')
    ,p_business_group_id        => 23
    ,p_last_name                => 'Bloggs'
```

```
    ,p_sex                          => 'M'
    ,p_person_type_id               => 56
    ,p_email_address                => 'bloggsf@uk.uiq.com'
    ,p_employee_number              => l_emp_num
    ,p_person_id                    => l_person_id
    ,p_assignment_id                => l_assignment_id
    ,p_per_object_version_number => l_per_object_version_number
    ,p_asg_object_version_number => l_asg_object_version_number
    ,p_per_effective_start_date  => l_per_effective_start_date
    ,p_per_effective_end_date    => l_per_effective_end_date
    ,p_full_name                    => l_full_name
    ,p_per_comment_id               => l_per_comment_id
    ,p_assignment_sequence          => l_assignment_sequence
    ,p_assignment_number            => l_assignment_number
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    ,p_orig_hire_warning         => l_orig_hire_warning
    );
    --
    -- The l_emp_num variable is now set with the
    -- employee_number allocated by the HR system.
    --
end;
```

### スタイル更新 API のデフォルト・パラメータ

スタイル更新 API の場合、主キーおよびオブジェクト・バージョン番号パラメータは通常必須です。たいていの場合、必ずしもすべてのパラメータ値を指定する必要はありません。指定する必要があるのは、いずれかの制御パラメータと実際に変更している属性のみです。変更していない属性の既存の値を渡すことも可能ですが、必ずしも必要ではありません。オプション・パラメータは、データ型によって次の PL/SQL デフォルト値のうちの 1 つになります。

| データ型 | デフォルト値 |
|---|---|
| varchar2 | hr_api.g_varchar2 |
| number | hr_api.g_number |
| date | hr_api.g_date |

前述の hr_api.g_ デフォルト値は、定数定義で、特殊値に設定されています。これらのデフォルト値は、ハードコードされるテキスト文字列ではありません。これらの値を指定する必要がある場合には、値ではなく、定数名を使用してください。実効値は変更されることがあります。

IN OUT パラメータはコール側パラメータ・リストに必ず含める必要があるので、注意が必要です。API は値を渡すことができるので、この種のパラメータには変数を使用して値を渡す必要があります。これらの変数には、API をコールする前にユーザー定義の値を指定してください。属性を明示的に変更しない場合には、この変数をそのデータ型の hr_api.g_…値

に設定します。「update_emp_asg_criteria」API には、こうした各種パラメータの例が含まれています。

```
procedure update_emp_asg_criteria
(...
,p_assignment_id in number
,p_object_version_number in out number
...
,p_position_id in number
                              default hr_api.g_number
...
,p_special_ceiling_step_id in out number
...
,p_employment_category in varchar2
                              default hr_api.g_varchar2
,p_effective_start_date out date
,p_effective_end_date out date
,p_people_group_id out number
,p_group_name out varchar2
,p_org_now_no_manager_warning out boolean
,p_other_manager_warning out boolean
,p_spp_delete_warning out boolean
,p_entries_changed_warning out varchar2
,p_tax_district_changed_warning out boolean
);
```

**注意：** 特に重要なパラメータのみが示されています。省略記号（...）は、この例と無関係のパラメータが省略されていることを意味します。

p_assignment_id と p_object_version_number の 2 つのパラメータは必須なので、すべてのコールに指定する必要があります。p_position_id パラメータはオプションです。既存の値を変更しない場合には、コール側ロジックからこのパラメータを除外するか、hr_api.g_varchar2 定数か既存の値を渡します。

p_special_ceiling_step_id パラメータは IN OUT パラメータです。いくつかのケースに関しては、API はデータベースのこの属性を NULL に設定します。API からは最新の値が渡されます。この属性を変更しない場合には、コール側ロジック変数を hr_api.g_number に設定します。

### 例

次に、position_id 属性や special_ceiling_step_id 属性は変更しないが、employment_category 値は変更する場合の、「update_emp_asg_criteria」API のコール例を示します。

```
declare
  l_assignment_id                 number;
  l_object_version_number         number;
  l_special_ceiling_step_id       number;
  ...
begin
  l_assignment_id           := 23121;
  l_object_version_number   := 4;
  l_special_ceiling_step_id := hr_api.g_number;
  hr_assignment_api.update_emp_asg_criteria
    (...
    ,p_assignment_id                => l_assignment_id
    ,p_object_version_number        => l_object_version_number
    ...
    ,p_special_ceiling_step_id      => l_special_ceiling_step_id
    ...
    ,p_employment_category          => 'FT'
    ...
    );
  --
  -- As p_special_ceiling_step_id is an IN OUT parameter the
  -- l_special_ceiling_step_id variable is now set to the same
  -- value as on the database. i.e. The existing value before
  -- the API was called or the value which was derived by the
  -- API.The variable will not be set to hr_api.g_number.
  --
end;
```

### スタイル削除 API のデフォルト・パラメータ

ほとんどのスタイル削除 API は、いずれの属性パラメータに関してもデフォルト値を持ちません。まれに、デフォルト値を持つパラメータがスタイル更新 API のそうしたパラメータと同様の働きをすることはあります。

## p_validate 制御パラメータについて

すべての公開済 API に p_validate 制御パラメータが含まれています。このパラメータが FALSE（デフォルト値）に設定されている場合には、プロシージャはその業務機能の全検証を実行します。この操作が有効な場合には、データベースの行 / 値が挿入、更新、または削除されます。すべての非警告 OUT パラメータ、警告 OUT パラメータ、IN OUT パラメータに、特定の値が指定されます。

p_validate パラメータが TRUE に設定されている場合には、API は操作が有効かどうかのみをチェックします。これは、プロシージャの始めにセーブポイントを発行し、最後にこのセーブポイントにロールバックすることによって行われます。ユーザーがこうした内部セーブポイントにアクセスすることはできません。プロシージャが検証エラーを一切引き起こすことなく正常に完了した場合には、非警告 OUT パラメータは NULL に、警告 OUT パラメータは特定の値にそれぞれ設定されます。また、IN OUT パラメータは IN 値にリセットされます。

場合によっては、パブリック API プロシージャを基本ブロックとして使用して PL/SQL ルーチンを記述することもできます。これにより、ビジネス・ニーズに合った固有のルーチンを記述することができます。たとえば、業務上、デートトラックを任意の行に適用し、同一行に先日付デートトラック削除を適用します。このような場合には、標準 API のうちの 2 つをコールする「update_and_future_del」プロシージャを記述することができます。

各標準 API をコールするときは、p_validate を FALSE に設定してください。TRUE を使用した場合には、更新プロシージャ・コールがロールバックされます。このため、削除プロシージャがコールされたときに、更新されていない行が影響を受けます。しかし p_validate が FALSE に設定されていれば、更新はロールバックされません。このため、削除コールはユーザーがトランザクション全体の適用を意図したかのように実行されます。

更新および削除操作が有効であるかチェックする場合には、独自のセーブポイントとロールバック・コマンドを発行する必要があります。API がコミットを発行することはないため、作業の一部がデータベース内に残される危険性はありません。コミットの発行は、コール側のコードが行う必要があります。次では、p_validate の TRUE 動作の一部をシミュレートします。

**例**

```
savepoint s1;
update_api_prc(.........);
delete_api_prc(..........);
rollback to s1;
```

セーブポイント名にシステム定義の API プロシージャ名を使用しないでください。別の名前を使用しないと、予期せぬ結果が生じる恐れがあります。

## p_effective_date 制御パラメータについて

少なくとも 1 つのデートトラック・エンティティのデータの挿入 / 更新 / 削除を行う API の大半に、p_effective_date 制御パラメータがあります。この必須パラメータは、操作の適用開始日を定義します。このパラメータの PL/SQL データ型は日付です。

デートトラックの最小時間単位は 1 日なので、p_effective_date パラメータの時間部分は使用されません。つまり、変更は必ず午前 0 時を境に有効になります。

API の中には、さらに明確な処理の日付を持つものもあります。たとえば、「create_employee」API には p_effective_date パラメータはありません。p_hire_date パラメータは、個人詳細が有効になる最初の日として用いられます。

**例 1**
この例では、今日から開始する新規等級レートを作成します。

```
hr_grade_api.create_grade_rate_value
(...
,p_effective_date => trunc(sysdate)
 ...);
```

**例 2**
この例では、1997 年 3 月初めに入社する新規従業員を作成します。

```
hr_employee_api.create_employee
(...
,p_hire_date => to_date('01-03-1997','DD-MM-YYYY')
...);
```

デートトラック・エンティティのデータを変更することのない API の中にも、p_effective_date パラメータを持つものがあります。変更がいつ有効となるかを決定するために日付値が使用されることはありません。日付値は、参照値の妥当性検査に使用されます。各参照値には、有効な日付範囲を指定できます。開始日は、いつその値を初めて使用することができるかを表します。終了日は、その値を新規レコードで使用することのできる、レコードの更新時に設定される最終日を表します。変更されない既存のレコードは、終了日以降も参照を継続して使用できます。

## p_datetrack_update_mode 制御パラメータについて

少なくとも 1 つのデートトラック・エンティティのデータを更新する API の大半に、p_datetrack_update_mode 制御パラメータがあります。このパラメータを使用すると、実行するデートトラック変更のタイプを定義することができます。この必須パラメータは、次のいずれかの値に設定してください。

| 値 | 説明 |
|---|---|
| UPDATE | 既存の履歴情報を保持する |
| CORRECTION | 既存の情報を訂正する |
| UPDATE_OVERRIDE | すべての変更予定を置換する |
| UPDATE_CHANGE_INSERT | 次の変更予定の前にこの変更を挿入する |

あらゆるケースであらゆるモードを使用することは不可能な場合があります。たとえば、変更しているレコードに既存の先日付変更がない場合には、UPDATE_OVERRIDE と UPDATE_CHANGE_INSERT の 2 つのデートトラック・モードは使用できません。

デートトラック・エンティティを更新する一部の API については、p_datetrack_update_mode パラメータがありません。こうした API は、その業務処理に関してデートトラック操作を自動的に実行します。

同じ主キーの日付付きインスタンスはそれぞれ、異なる object_version_number を持ちます。API をコールするときには、p_object_version_number パラメータを操作日（すなわち、p_effective_date）時点で適用する値に設定してください。

### 例

次の等級レート値がすでに pay_grade_rules_f 表に存在しているものと仮定します。

| Grade_rule_id | Effective Start_Date | Effective_ End_Date | version_ Number | Value |
|---|---|---|---|---|
| 12122 | 01-JAN-1996 | 20-FEB-1996 | 2 | 45 |
| 12122 | 21-FEB-1996 | 20-JUN-1998 | 3 | 50 |

さらに、等級レート値が 1996 年 2 月 21 日に誤った値に更新されていると仮定します。45 から 55 に更新するものが 45 から 50 に更新されているため、エラーを修正することにします。

```
declare
  l_object_version_number number;
  l_effective_start_date  date;
  l_effective_end_date    date;
begin
  l_object_version_number := 3;
  hr_grade_api.update_grade_rate_value
    (p_effective_date      => to_date('21-02-1996','DD-MM-YYYY')
    ,p_datetrack_update_mode => 'CORRECTION'
    ,p_grade_rule_id         => 12122
    ,p_object_version_number => l_object_version_number
    ,p_value                 => 55
    ,p_effective_start_date  => l_effective_start_date
    ,p_effective_end_date    => l_effective_end_date
    );
  -- l_object_version_number will now be set to the value
  -- as on database row, as of 21st February 1996.
end;
```

## p_datetrack_delete_mode 制御パラメータについて

少なくとも 1 つのデートトラック・エンティティのデータを削除する API の大半に、p_datetrack_delete_mode 制御パラメータがあります。このパラメータを使用すると、実行するデートトラック削除のタイプを定義することができます。この必須パラメータは、次のいずれかの値に設定してください。

| p_datetrack_delete_mode 値 | 説明 |
|---|---|
| ZAP | データベースから完全に削除する |
| DELETE | 終了日を有効日に設定する |
| FUTURE_CHANGE | すべての変更予定を削除する |
| DELETE_NEXT_CHANGE | 次の変更を削除する |

あらゆるケースであらゆるモードを使用することは不可能な場合があります。たとえば、変更しているレコードに既存の先日付変更がない場合には、FUTURE_CHANGE と DELETE_NEXT_CHANGE の 2 つのデートトラック・モードは使用できません。デートトラック・エンティティを更新する一部の API については、p_datetrack_delete_mode パラメータがありません。こうした API は、その業務処理に関してデートトラック操作を自動的に実行します。詳細は、各 API パッケージ・ヘッダー作成スクリプトのコメントで参照してください。

同じ主キーの日付付きインスタンスはそれぞれ、異なる object_version_number を持ちます。API をコールするときには、p_object_version_number パラメータを操作日（すなわち、p_effective_date）時点で適用する値に設定してください。

**例**

次の等級レート値がすでに pay_grade_rules_f 表に存在しているものと仮定します。

| Grade_rule_id | Effective_ Start_Date | Effective_ End_Date | Object_ Version_ Number | Value |
|---|---|---|---|---|
| 5482 | 15-JAN-1996 | 23-MAR-1996 | 4 | 10 |
| 5482 | 24-MAR-1996 | 12-AUG-1996 | 8 | 20 |

さらに、この等級レート値の日付付きインスタンスをデータベースからすべて削除すると仮定します。

```
declare
  l_object_version_number number;
  l_effective_start_date  date;
  l_effective_end_date    date;
begin

  l_object_version_number := 4;

  hr_grade_api.update_grade_rate_value
    (p_effective_date      => to_date('02-02-1996','DD-MM-YYYY')
    ,p_datetrack_delete_mode => 'ZAP'
    ,p_grade_rule_id         => 5482
    ,p_object_version_number => l_object_version_number
    ,p_effective_start_date  => l_effective_start_date
    ,p_effective_end_date    => l_effective_end_date
    );

  -- As ZAP mode was used l_object_version_number now is null.
end;
```

## p_effective_start_date および p_effective_end_date パラメータについて

少なくとも 1 つのデートトラック・エンティティのデータの挿入 / 更新 / 削除を行う API の大半に、p_effective_start_date と p_effective_end_date の 2 つの制御パラメータがあります。

これらのパラメータはいずれも、OUT と定義されます。

戻り値は、p_effective_date 時点で効力を生じる行の effective_start_date と effective_end_date データベース列の値に対応します。

こうしたパラメータは、特定の行のデートトラック・インスタンスがデータベースからすべて削除されたとき（すなわち、スタイル削除 API が ZAP デートトラック・モードでコールされたとき）に、NULL に設定されます。

### 例

次の等級レート値がすでに pay_grade_rules_f 表に存在しているものと仮定します。

```
                Effective_          Effective_
 Grade_rule_id  Start_Date          End_Date
 ---------------------------------------------
 17392          01-FEB-1996         24-MAY-1996
 17392          25-MAY-1996         01-SEP-1997
```

有効日 1996 年 3 月 10 日で UPDATE_CHANGE_INSERT デートトラック・モードを実行するために、「update_grade_rate_value」API がコールされます。API は、データベースの行を次のように変更します。

```
                Effective_          Effective_
 Grade_rule_id  Start_Date          End_Date
 ---------------------------------------------
 17392          01-FEB-1996         09-MAR-1996
 17392          10-MAR-1996         24-MAY-1996
 17392          25-MAY-1996         01-SEP-1997
```

API の p_effective_start_date パラメータが 10__MAR_1996 に、p_effective_end_date パラメータが 24_MAY_1996 にそれぞれ設定されます。

## p_language_code パラメータについて

p_language_code パラメータは、スタイル作成およびスタイル更新複数言語サポート API でのみ使用可能です。このパラメータを使用すると、翻訳値に適用する言語を指定できます。このパラメータは基本言語またはインストール済の任意の言語に設定できます。このパラメータの hr_api.userenv_lang というデフォルト値は、次の指定に相当します。

```
select userenv('LANG')
    from dual;
```

このパラメータを NULL または hr_api.g_varchar2 に設定した場合にも、hr_api.userenv_lang というデフォルトが使用されます。

参照 : 複数言語サポート : 12-24 ページ

# API 機能

## コミット文

コミットを発行する人事管理システム API は 1 つもありません。コミット文の発行は、コール側のコードが行う必要があります。このため、トランザクションの一部がデータベース内に残されることは絶対にありません。エラーが生じれば、すべてのトランザクションはロールバックされます。したがって、API 作業は完了するか、まったく行われないかどちらかになります。人事管理システム API を「基本ブロック」として使用することにより、独自の業務機能を構築することができます。このため、必要に応じて自由にコミットを発行することができます。

また、異なるクライアント・ツールとのコンフリクトも回避されます。たとえば、Oracle Forms は、ユーザーの変更にエラーがまったくない場合にのみコミットを発行します。これは、1 つ以上のレコード変更、おそらく個別の API コールになるものと思われます。

## デッドロックの回避

同一のコミット単位内の 2 つ以上の API をコールする場合は、デッドロック状況が発生しないように注意してください。表ロック・ラダーにリストされている順番に表にアクセスすれば、デッドロックは回避されるはずです。たとえば、表内の「処理順序」が最下位の行からまず更新または削除してください。

同じ表内の 2 つ以上の行にアクセスしている場合には、昇順の主キー順序で行をロックします。たとえば、1 個人の全アサイメントを更新している場合は、assignment_id が最も小さい行を最初に変更します。

操作をロック・ラダー順に実行することが不可能な場合には、明示的なロック・ロジックが必要とされます。表が処理順に表示される場合は、ジャンプし、後でアクセスする表の行はすべて、あらかじめ明示的にロックする必要があります。表をジャンプし、更新または削除する行がその表に 1 つもない場合には、その表には一切ロックをかけないでください。

### 例

ロック・ラダー順序が次のとおりであると仮定します。

```
Table      Processing Order
  A             10
  B             20
  C             30
  D             40
```

さらに、ロジックが次の順序で行を更新する必要があると仮定します。

```
A 1st
D 2nd
C 3rd
```

したがって、ロジックは次のようになる必要があります。

1. 表 A の行を更新する。
2. 表 C の行をロックする（ステップ 4 で更新される予定の行のみをロックします）。
3. 表 D の行を更新する。
4. 表 C の行を更新する。

表 B は、D の後にアクセスされないのでロックされません。ロックは API において最初のステップの一環として行われるため、コードが表 A または D の行を明示的にロックする必要はありません。

つまり、更新または削除の順序はユーザーが選択できますが、表の行は、表ロック・ラダーに示された順序でロックする必要があります。

## API のフレックスフィールド

API は、Oracle アプリケーション・オブジェクト・ライブラリ・フォームを使用して作成されたフレックスフィールドの定義を使用して、付加フレックスフィールドおよびキー・フレックスフィールドの列値を検証します。

API のフレックスフィールド検証はデータベース内で実行されるため、値セットの定義では、フィールドなど、フォーム・オブジェクトを直接に参照することはできません。サーバー側検証ではこれらの参照を解決できないため、すべてのチェックが失敗します。サーバー側ではプロファイル値を使用できないこともあるため、プロファイルを参照するときにも注意が必要です。

コミット時処理のためにフォームが現在 API をコールしていないとしても、値セット定義ではフォーム・フィールドを直接に参照しないようにお薦めします。将来アップグレードしたときに問題が発生する可能性があります。他のフィールド検証を実行する場合や、値セットにインプリメントできないフレックスフィールド検証を実行する場合には、API ユーザー・フックを使用してください。

参照 : API ユーザー・フック : 12-32 ページ

フレックスフィールド検証に関連する問題の情報、および問題の解決方法については、次の項を参照してください。フレックスフィールド値の検証 : 10-1 ページ

API はフレックスフィールド値のセキュリティを強制施行しません。これは、フォーム・ユーザー・インタフェースを使用する場合にのみ行うことができます。

Oracle Applications では、各付加フレックスフィールドに構造列が定義されています。多くの場合、構造列名は文字で終わります。つまり、「ATTRIBUTE_CATEGORY」と名付けられています。インプリメンテーション・チームはこの構造列と参照フィールドを関連付けることができます。構造列の値は、どのフレックスフィールド体系を検証対象とするかに影響を与えることができます。参照フィールドが定義されている場合に API をコールするには、ATTRIBUTE_CATEGORY 値に参照フィールド値を挿入する必要があります。

付加フレックスフィールドの場合、API は通常、現行表のその他の列の検証後にフレックスフィールド検証を実行します。キー・フレックスフィールド・セグメントの場合、値は組合せ表と呼ばれる別個の表に入れられます。行はメイン製品表の前に組合せ表に保持されるため、API はメイン製品表の列を検証する前にフレックスフィールド検証を実行します。

リリース 11.0 以前は、API でフレックスフィールド検証を実行する前に、スケルトン・フレックスフィールド検証パッケージ本体作成スクリプトのコピーを編集する必要がありました。このような処理を必要としていた技術上の制約は解消されました。現在はこれらのスケルトン・ファイル、*fli.pkb は製品に搭載されていません。

## 複数言語サポート

人事管理システム・スキーマ中のいくつかのエンティティでは、複数言語サポート（MLS）が提供されています。翻訳値は _TL 表に保持されます。MLS 概念の一般的な情報は、次の資料で参照してください。

> **関連項目：** MLS の原則については Oracle Applications の概念のマニュアル、MLS の構成については Oracle Applications のインストール・ガイド。

翻訳値と未翻訳値は同じ代理キー ID の列および値で識別されるため、複数言語サポート API では、両方のグループの値が同じ PL/SQL プロシージャ・コールで管理されます。

スタイル作成およびスタイル更新の API には、翻訳値に適用する言語を指定するために使用する p_language_code パラメータがあります。API は必要な行を _TL 表に保持し、source_lang および language 列を適切に設定します。これらの列、および p_language_code パラメータは、FND_LANGUAGES 表からの language_code 値を保持します。

p_language_code パラメータには、次の指定に相当する hr_api.userenv_lang というデフォルト値があります。

```
select userenv('LANG')
  from dual;
```

p_language_code パラメータを設定することにより、異なる言語の翻訳データを同じデータベース・セッション内で維持できます。このパラメータを NULL または hr_api.g_varchar2 に設定した場合にも、hr_api.userenv_lang というデフォルトが使用されます。

スタイル作成複数言語 API をコールした場合には、基本言語およびインストール済言語のそれぞれについて、_TL 表に行が挿入されます。それぞれの行で、source_lang 列は p_language_code パラメータと等しく、翻訳された列の値は同じになります。他の翻訳値を使

用できる場合には、p_language_code パラメータを適切な言語コードに設定して、更新 API をコールすることによりこれらの値を設定できます。

スタイル更新複数言語サポート API をコールするたびに、未翻訳の値と、一組の翻訳値を更新できます。API はメイン表の中の未翻訳の値と、_TL 表の中の、対応する行にある翻訳データ値を更新します。p_language_code パラメータが language または source_lang 列と一致している行で、翻訳列が更新されます。source_lang 列に対する照合を含めることにより、明示的に設定されていない翻訳と、作成された言語の同期を保つことができます。最初に翻訳を設定したときには、source_lang 列も p_language_code の値で更新されます。他の翻訳用に値を変更する場合は、p_language_code および翻訳パラメータを適切に設定して、再度更新 API をコールしてください。

スタイル削除複数言語サポート API の場合には、p_language_code パラメータはありません。未翻訳データを削除すると、_TL 表の中の対応する翻訳行もすべて削除されます。このため、API は特定の言語に関して処理を実行する必要はありません。

複数言語 API をコールした場合、_TL 表の中の複数の行が処理される可能性があります。変更されるすべての行を識別することを避けるため、_TL 表には object_version_number 列はありません。未翻訳値を保持しているメイン表には、object_version_number 列があります。複数言語サポート API を使用するときには、翻訳値の更新のみを行う場合にでも、メイン表の値を使用して p_object_version_number パラメータを設定してください。

## 代替 API

時には、2 つ以上の API を使用して同じビジネス・プロセスを実行することも可能です。これは特に、複数のエンティティが各種国別仕様の追加詳細を保持している場合に当てはまります。一般的には、いずれの国別仕様にも、また一部の国別仕様の特定バージョンにも使用可能なメイン API があります。いずれの API がコールされた場合でも、同様の検証と変更がデータベースに対して行われます。

たとえば、個人情報の住所を保持するエンティティがあります。UK 形式の住所の場合、特定詳細の保持にはいくつかの一般住所属性が使用されます。

```
PER_ADDRESSES         create_person         create_gb_person
Table                 _address API          _address API
Column Name           Parameter Name        Parameter Name
-------------------------------------------------------------
style                 p_style               N/A
address_line1         p_address_line1       p_address_line1
address_line2         p_address_line2       p_address_line2
address_line3         p_address_line3       p_address_line3
town_or_city          p_town_or_city        p_town
region_1              p_region_1            p_county
region_2              p_region_2            N/A for this style
region_3              p_region_3            N/A for this style
postal_code           p_postal_code         p_postcode
country               p_country             p_country
telephone_number_1    p_telephone_number_1  p_telephone_number
```

```
telephone_number_2    p_telephone_number_2  N/A for this style
telephone_number_3    p_telephone_number_3  N/A for this style
```

**注意：** すべてのデータベース列名または API パラメータがリストされているわけではありません。

p_style パラメータは 1 つの形式の住所のみを作成するため、「create_gb_person_address」API には存在していません。

すべての住所属性がすべての形式で使用されるわけではありません。たとえば、region_2 属性を UK 形式の住所に設定することはできません。したがって、「create_gb_person_address」API には対応するパラメータはありません。p_style を "GB" に設定して「create_person_address」API をコールするときは、p_region_2 は NULL にする必要があります。

最高の柔軟性を得るために、両方のインタフェースが提供されています。事業所が 1 箇所にしかない場合には、自国に合った住所形式のインタフェースをコールした方が便利でしょう。事業所が何箇所もあり、地域別形式で住所詳細を保存する場合には、一般 API をコールし、作成時に希望する形式を指定した方が便利でしょう。

その他の代替インタフェースについての詳細は、各 API パッケージ・ヘッダー作成スクリプトのコメントを参照してください。

参照 : ユーザー・フックおよび代替インタフェース API: 12-52 ページ

# API エラーと警告

## 障害エラー

API のコール時には、検証エラーまたは処理エラーが発生する可能性があります。こうしたエラーは、Oracle Applications における他の PL/SQL エラーと同じように引き起こされます。

エラーが発生した場合には、その単一の API コールによって実行された作業全体がロールバックされます。API がコミットを発行することはないので、作業の一部がデータベース内に残される危険性はありません。コミットの発行は、コール側のコードが行う必要があります。

## 警告値

警告は OUT パラメータを使用して返されます。こうしたパラメータの名前は、_WARNING で終わります。たいていの場合、データ型はブールです。警告値が発生すると、このパラメータは TRUE に設定されます。データ型がブールでない場合は、他の値が返されます。詳細は、各 API パッケージ・ヘッダー作成スクリプトのコメントで参照してください。

API は、警告状況にフラグが立てられてはいますが、引き続き受け入れ可能であると見なします。深刻なデータ問題の危険性があった場合には、PL/SQL エラーが引き起こされ、現在の API コールの処理は停止されます。

しかし、特定の組織においては、警告内容を記録したり、詳細なチェックを行う必要が場合によってはあります。変更が行われている間は変更内容をデータベース内に保持しない場合には、API が実行した作業を明示的にロールバックする必要があります。

**例**

「create_employee」API がコールされた場合、すでにデータベース内にある個人詳細に同一の last_name、first_name、date_of_birth の組合せがあるときには、*p_name_combination_warning* パラメータは TRUE に設定されます。

```
declare
  l_name_combination_warning   boolean;
  l_assign_payroll_warning     boolean;
begin
  savepoint on_name_warning;
  hr_employee.create_employee
    (p_validate           => false
    ...
    ,p_last_name                 => 'Bloggs'
    ,p_first_name         => 'Fred'
    ,p_date_of_birth      => to_date('06-06-1964', 'DD-MM-YYYY')
    ...
    ,p_name_combination_warning  => l_name_combination_warning
    ,p_assign_payroll_warning    => l_assign_payroll_warning
    );
  if l_name_combination_warning then
    -- Note that similar person details already exist.
    -- Do not hold the details in the database until it is
    -- confirmed this is really a different person.
    rollback to on_name_warning;
  end if;
end;
```

**注意：** p_validate パラメータが TRUE に設定されている場合には、API 作業をロールバックする必要はありません。

セーブポイント名にシステム定義の API プロシージャ名を使用しないでください。別の名前を使用しないと、予期せぬ結果が生じる恐れがあります。

### PL/SQL バッチ処理におけるエラー処理

バッチ環境においては、バッチ処理に対して提起されたエラーを処理し、記録して、処理を続行できるようにする必要があります。こうしたバッチ処理の企画を支援するために、HR_API_BATCH_MESSAGE_LINES と呼ばれるメッセージ表およびいくつかの API が提供されています。次を参照してください。

| API 名 | 説明 |
|---|---|
| create_message_line | HR_API_BATCH_MESSAGE_LINES 表に単一のエラー・メッセージを追加する。 |
| delete_message_line | HR_API_BATCH_MESSAGE_LINES 表の単一のエラー・メッセージを削除する。 |
| delete_message_lines | ある特定のバッチ実行の全エラー・メッセージ・ラインを削除する。 |

各 API の詳しい説明は、パッケージ・ヘッダー作成スクリプトのコメントを参照してください。

PL/SQL バッチ処理における API エラーを処理するために、HR_API_BATCH_MESSAGE_LINES 表にメッセージを保存することをお薦めします。

## PL/SQL バッチ・プログラム例

従業員の住所が含まれている一時表が作成されたと仮定します。これらの住所は、人事システム・スキーマに挿入する必要があります。住所が保持されている一時表は、temp_person_address と呼ばれます。Sql*Loader を使用すれば、ASCII ファイルから挿入することができたでしょう。

TEMP_PERSON_ADDRESSES 表

| 列名 | データ型 |
|---|---|
| person_id | number |
| primary_flag | varchar2 |
| date_from | date |
| address_type | varchar2 |
| address_line1 | varchar2 |
| address_line2 | varchar2 |
| address_line3 | varchar2 |
| town | varchar2 |
| county | varchar2 |
| postcode | varchar2 |
| country | varchar2 |
| telephone_number | varchar2 |

### サンプル・コード

```
declare
  --
  l_rows_processed  number := 0; -- rows processed by api    l_commit_point
number := 20; - Commit after X successful rows
  l_batch_run_number hr_api_batch_message_lines.batch_run_number%type;
  l_dummy_line_id        hr_api_batch_message_lines.line_id%type;    l_address_id
per_addresses.address_id%type;
  l_object_version_number_id per_addresses.object_version_number_id%type;
  --
  -- select the next batch run number
  --
  cursor csr_batch_run_number is
    select nvl(max(abm.batch_run_number), 0) + 1
      from hr_api_batch_message_lines abm;
  --
  -- select all the temporary 'GB' address rows
  --
  cursor csr_tpa is
  select tpa.person_id
       , tpa.primary_flag
       , tpa.date_from
       , tpa.address_type
       , tpa.address_line1
       , tpa.address_line2
       , tpa.address_line3
       , tpa.town
       , tpa.county
       , tpa.postcode
       , tpa.country
       , tpa.telephone_number
       , tpa.rowid
  from temp_person_addresses tpa
  where tpa.address_style = 'GB';
begin
  -- open and fetch the batch run number
  open csr_batch_run_number;
  fetch csr_batch_run_number into l_batch_run_number;
  close csr_batch_run_number;
  -- open and fetch each temporary address row
  for sel in csr_tpa loop
    begin
      -- create the address in the HR Schema
      hr_person_address_api.create_gb_person_address
        (p_person_id             => sel.person_id
        ,p_effective_date        => trunc(sysdate)
        ,p_primary_flag          => sel.primary_flag
```

```
      ,p_date_from             => sel.date_from
      ,p_address_type          => sel.address_type
      ,p_address_line1         => sel.address_line1
      ,p_address_line2         => sel.address_line2
      ,p_address_line3         => sel.address_line3
      ,p_town                  => sel.town
      ,p_county                => sel.county
      ,p_postcode              => sel.postcode
      ,p_country               => sel.country
      ,p_telephone_number      => sel.telephone_number
      ,p_address_id            => l_address_id
      ,p_object_version_number => l_object_version_number
      );
    -- increment the number of rows processed by the api
    l_rows_processed := l_rows_processed + 1;
    -- determine if the commit point has been reached
    if (mod(l_rows_processed, l_commit_point) = 0) then
      -- the commit point has been reached therefore commit
      commit;

    end if;
 exception
 when others then
       --
   -- An API error has occurred
   -- Note: As an error has occurred only the work in the
   -- last API call will be rolled back. The
   -- uncommitted work done by previous API calls will not be
   -- affected. If the error is ora-20001 the fnd_message.get
   -- function will retrieve and substitute all tokens for
   -- the short and extended message text. If the error is
   -- not ora-20001, null will be returned.
   --
   hr_batch_message_line_api.create_message_line
     (p_batch_run_number => l_batch_run_number
     ,p_api_name               =>
'hr_person_address_api.create_gb_person_address'
     ,p_status => 'F'
     ,p_error_number => sqlcode
     ,p_error_message => sqlerrm
     ,p_extended_error_message => fnd_message.get
     ,p_source_row_information => to_char(sel.rowid)
     ,p_line_id => l_dummy_line_id);
   end;
 end loop;
 -- commit any final rows
 commit;
end;
```

次のように、完了したバッチ実行の HR_API_BATCH_MESSAGE_LINES 表から選択することによって、処理の実行中に作成された可能性のあるエラーをすべて表示することができます。

```
select *
  from hr_api_batch_message_lines abm
 where abm.batch_run_number = :batch_run_number
 order by abm.line_id;
```

## WHO 列と Oracle Alert

Oracle Applications の表の多くには標準 WHO 列があります。次のものが含まれます。

- LAST_UPDATE_DATE
- LAST_UPDATED_BY
- LAST_UPDATE_LOGIN
- CREATED_BY
- CREATION_DATE

これらの列に保持されている値は、通常、そのデータベースの行を作成または更新させたアプリケーション・ユーザーを表します。オラクル人事管理システム ・アプリケーションでは、これらの列はデータベース・トリガーによって保守されます。対応する API パラメータが提供されていないので、ユーザーがこれらの列に直接挿入することはできません。

API が Application Form またはコンカレント・マネージャ・セッションから実行された場合には、その Form がデータベース変更を実行したかのように、これらの列は保守されます。

API が SQL*Plus データベース・セッションからコールされた場合には、CREATION_DATE 列と LAST_UPDATE_DATE 列には依然としてデータベースの *sysdate* 値が挿入されます。アプリケーション・ユーザー詳細がないため、CREATED_BY、LAST_UPDATED_BY、LAST_UPDATE_LOGIN の各列は「無名ユーザー」値に設定されます。

SQL*Plus データベース・セッションで CREATED_BY 列と LAST_UPDATED_BY 列に既知のアプリケーション・ユーザーの詳細を挿入する場合には、人事管理システム API を実行する前に必ず、次のサーバー側パッケージ・プロシージャを 1 回コールしてください。

```
fnd_global.apps_initialize
```

このプロシージャをコールする場合には、誤った値は拒否されませんので、有効な値を渡してください。誤った値は拒否されます。このプロシージャは、Oracle Alert と API を使用する場合にもコールする必要があります。

AOL プロファイルを使用すれば、HR セキュリティ・プロファイルと AOL 職責を関連付けることができます。apps_initialize resp_id パラメータを制限付き HR セキュリティ・プロ

ファイルに関連付けられた職責に設定する場合には、注意が必要です。API 検証が過度に制限されないようにするには、その職責のビジネス・グループ内に保持されているデータの保守が欠かせません。

同じデータベース・セッションで 2 つ以上のビジネス・グループのデータを保守するには、無制限 HR セキュリティ・プロファイルと関連付けられた職責を使用します。

## API ユーザー・フック

オラクル人事管理システムの API は、カスタム・ビジネス・ロジックの追加をサポートしています。オラクル社ではこの機能を「API ユーザー・フック」と呼んでいます。こうしたフックを使用することにより、API が実行する標準ビジネス・ルールを拡張できます。独自の検証ルールまたは他の処理ロジックを含めて、関連する API が実行される度に自動的に実行されるようにすることができます。

次の事柄について検討してください。

- 顧客固有のデータの妥当性検査

  たとえば、従業員の昇格・昇進時には、特定の事業所に勤務しているか、6 か月以上その等級であった場合を除いて、必要に応じて等級の変更をシングル・ステップに制限することができます。

- 追加の顧客固有の表に保持されているデータの保守

  たとえば、Oracle Applications にないデータベース表に、特定の市場情報または従業員に関する評価情報を必要に応じて保存することができます。

- 特定のビジネス・イベントが発生したという事実の捕捉

  たとえば、必要に応じて、ある従業員が退職するという事実を捕捉して、個別のセキュリティ・データベースに直接電子メッセージを送信することによって、その従業員のオフィス・セキュリティ・パスを使用禁止にすることができます。

ユーザー・フックは、追加ロジックの実行が可能な API 内の場所です。API 処理がユーザー・フックに達すると、メイン処理が停止して、なんらかのカスタム・ロジックが実行されます。エラーが発生しなかった場合には、その後メイン API 処理が続行されます。

---

**警告：** Oracle に搭載されている API コード・ファイルは絶対に編集しないでください。これらのファイルはシステム定義の製品コードの一部であるため、修正した場合には、お客様の導入をサポート / アップグレードできない場合があります。Oracle Applications では、公開済 API の直接コールのみサポートします。オラクル人事管理システム製品セットの一部として記述された他のサーバー側パッケージ・プロシージャまたは関数の直接コールは、特に明記されていない限りサポートされていません。

---

## API ユーザー・フックのインプリメント

API と関連付けたい追加ロジックはすべて、PL/SQL を使用して個別のサーバー側パッケージ・プロシージャとしてインプリメントする必要があります。ビジネス・ルール・モデルの分析および設計は、お客様の導入に固有のものです。この項では、記述するルールと API ユーザー・フックの関連付け方法について説明します。

ルールを記述し、データベースのサーバー側パッケージにロードしたら、パッケージを 1 つ以上の特定のユーザー・フックと関連付ける必要があります。こうした情報を挿入、更新および削除する 3 つの特別 API があります。システム定義の API と追加ロジックの間のリンクを作成するには、システム定義のプリプロセッサ・プログラムを実行します。これによりユーザーが定義したデータが確認され、コールするパッケージ・プロシージャがチェックされ、特定のユーザー・フックから PL/SQL を実行するためのロジックが構築されます。このステップは、ユーザー・フックによる API 実行の総合的なパフォーマンスを最適化するために提供されています。実際に、各 API は、明示的にチェックしなくても、実行する追加ロジックを認識します。

API と追加ロジックの間のリンクはデータに保持されるので、アップグレードを容易にサポートできます。同じ API ユーザー・フックとパラメータが新バージョンに存在している場合は、プリプロセッサ・プログラムを再度実行できます。このプロセスでは、Oracle Applications またはユーザー独自のソース・コード・ファイルを手動で編集しなくても、特定のユーザー・フックから PL/SQL を実行するために必要な追加コードが再構築されます。

## API ユーザー・フックをインプリメントするには

1. 追加ロジックを付加する API とユーザー・フックを指定します。参照 : 使用可能なユーザー・フック : 12-34 ページ
2. 使用するユーザー・フックで使用可能なデータ値を指定します。参照 : ユーザー・フックで使用可能なデータ値 : 12-38 ページ
3. 追加ロジックを PL/SQL サーバー側パッケージ・プロシージャにインプリメントします。参照 : 個別のパッケージ・プロシージャへの追加ロジックのインプリメント : 12-39 ページ
4. 「*hr_api_hook_call_api.create_api_hook_call*」API をコールして、追加 PL/SQL パッケージを適切な API ユーザー・フックに登録します。ユーザー・フックとサーバー側パッケージ・プロシージャの間のデータ・マッピングを定義します。参照 : カスタム・プロシージャのユーザー・フックへのリンク : 12-42 ページ
5. ユーザー・フック・プリプロセッサ・プログラムを実行します。これにより PL/SQL サーバー側パッケージ・プロシージャに対するそれらのパラメータの妥当性が検査され、別のパッケージ本体が動的にデータベース内に直接生成されます。生成されたコードには、API ユーザー・フックからカスタム・パッケージ・プロシージャをコールする PL/SQL が含まれます。参照 : API ユーザー・フック・プリプロセッサ・プログラム : 12-46 ページ

## 使用可能なユーザー・フック

API ユーザー・フックは、情報を作成、保守または削除する人事管理システム API に含まれています。たとえば、「create_employee」や「update_emp_asg_criteria」API が挙げられます。

**注意：** 代替インタフェース API にはユーザー・フックは提供されていません。たとえば、「create_us_employee」と「create_gb_employee」はいずれも、「create_employee」API の代替インタフェース API です。いかなる外部ロジックもメイン API に関連付ける必要があります。ユーザー・フックはまた、「create_message_line」などのユーティリティ形式の API にも含まれていません。

各種ユーザー・フックをすべてリストする PL/SQL スクリプトも使用できます。

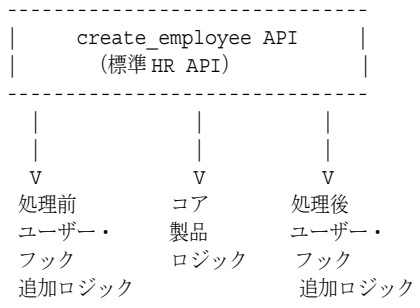
**関連項目：** API ユーザー・フック・サポート・スクリプト : 12-54 ページ

人事管理システム用のメイン API には、次の 2 種類のユーザー・フックがあります。

- 処理前
- 処理後

各 API には、これら 2 種類のユーザー・フックの異なるバージョンがあります。たとえば、「create_employee」API には「処理前」と「処理後」ユーザー・フックがあり、「update_person」API には別の「処理前」と「処理後」ユーザー・フックがあります。このため、ユーザー独自のロジックを特定の API とユーザー・フックにリンクすることができます。

### メイン API ユーザー・フック



### 処理前ロジック

処理前ユーザー・フックは、メイン API 処理ロジックがデータベース内のデータを処理する前に追加ロジックを実行します。この場合、検証の大半は実行されていないはずです。この種のユーザー・フックから追加ロジックをインプリメントする場合は、コンテキストとデータ値の妥当性はまったく検査されていないということを覚えておいてください。値は無効であり、メイン API 処理ロジックが実行されると拒否されるという可能性があります。

### 処理後ロジック

処理後ユーザー・フックは、メイン API 検証・処理ロジックがすべて完了した後に追加ロジックを実行します。API によって行われる予定のデータベース変更はすべて行われているはずです。また、これらのユーザー・フックから提供された値はすべて、妥当性チェックに合格しているはずです。追加検証では、提供された値が正しいと仮定することができます。エラーのためにメイン処理ロジックが完了しなかった場合には、「処理後」ユーザー・フックはコールされません。

---

**注意：** コア製品ロジックを変更することはできません。これは、「処理前」ユーザー・フックと「処理後」ユーザー・フックの間に実行されます。ユーザー・フックでは、追加カスタム・ロジックしか追加できません。

---

### コア製品ロジック

コア製品ロジックは、いくつかのコンポーネントに分けられます。API による変更が可能な表の場合は、内部行ハンドラ・コード・モジュールがあります。これらの行ハンドラは、API が使用可能なシステムのほとんどすべての表に関してインプリメントされています。内部行ハンドラは、メイン API が必要とする挿入、更新、削除、ロックのすべての処理を制御します。たとえば、メイン API は PER_ALL_PEOPLE_F 表に新規行を挿入する必要がある場合、自分自身で DML を実行することはありません。API はそのかわりに PER_ALL_PEOPLE_F 行ハンドラ・モジュールを実行します。

Oracle Applications は、こうした内部行ハンドラの直接コールは一切サポートしていません。これは、これらの内部行ハンドラには完全な検証・処理ロジックが含まれていないためです。サポートされている API と公開済 API のリストのコールのみが許可されています。このリストは『Oracle HRMS ユーザーズ・ガイド』およびオラクル人事管理システム・ヘルプの「パブリック・コールが可能なビジネス・プロセス API」のトピックに記載されています。リリースの新バージョンで採用された新規 API は、ヘルプ・システムの「オラクル人事管理システムの新機能」のトピックで紹介されます。

各行ハンドラ・モジュールでは、「挿入後」、「更新後」、「削除後」の 3 種類のユーザー・フックの使用が可能です。ユーザー・フック追加ロジックは、現在の表列に固有の検証が完了した後で、対応する表の DML 文の直後に実行されます。

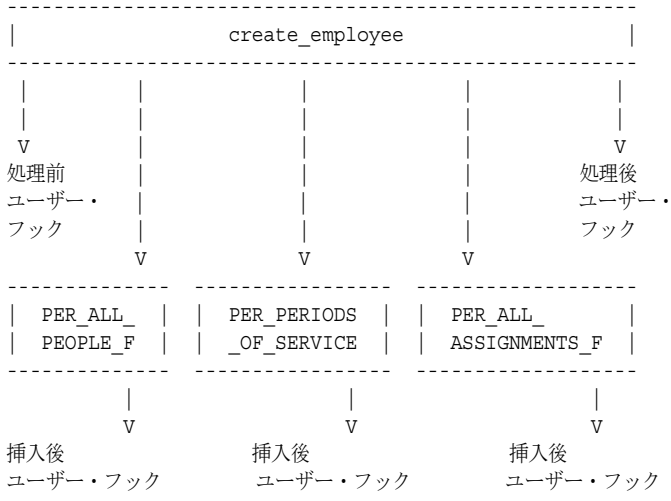
こうした行ハンドラ・ユーザー・フックは、次の 2 つの理由から、DML が完了した後に提供されます。

- すべてのコア製品検証が実行されている。このため、その特定の表に対する変更が有効であることがわかる。
- 挿入に関しては、主キー値は行が実際に挿入されるまではわからない。

> **注意：** DML 文の更新または削除が実行されたとしても、その前（DML の前）の列値はいずれのユーザー・フック・ロジックにも使用可能であることに変わりはありません。これについては、この項の後のセクションで詳細に説明します。

API が 2 つ以上の表にレコードを挿入、更新または削除するときには、ユーザーは多数のユーザー・フックを使用することができます。たとえば、「create_employee」API を使用して 6 つの異なる表の中にデータを作成できます。

**従業員作成 API のサマリー・コード・モジュール構造**




この図では、*create_employee* がサポートされている公開済 API です。内部行ハンドラのうちの *PER_ALL_PEOPLE_F*、*PER_PERIODS_OF_SERVICE* および *PER_ALL_ASSIGNMENTS_F* の 3 つのみが示されています。これらの内部行ハンドラは、直接コールしないでください。

ユーザー・フックの実行順序は次のとおりです。

1 番目）従業員作成 API 処理前ユーザー・フック

2 番目）PER_ALL_PEOPLE_F 行ハンドラ挿入後ユーザー・フック

3 番目）PER_PERIODS_OF_SERVICE 行ハンドラ挿入後ユーザー・フック

4 番目）PER_ALL_ASSIGNMENT_F 行ハンドラ挿入後ユーザー・フック

...

最後）従業員作成 API 処理後ユーザー・フック

---

**注意：** コア製品検証および処理ロジックは、ユーザー・フックとユーザー・フックの間に実行されます。

---

検証エラーまたは処理エラーが検出されると、PL/SQL 例外を引き起こして、処理がただちに中止されます。API 検証は、個々のコード・モジュールで実行されます。たとえば、「create_employee」API が使用されているときには、検証ロジックは実行される行ハンドラ 1 つ 1 つで実行されます。PER_PERIODS_OF_SERVICE 行ハンドラで妥当性チェック違反が発生したと仮定してみましょう。この場合、最初の 2 つのユーザー・フックに対して定義されているロジックが実行されます。PL/SQL 例外が引き起こされるため、その API コールの 3 番目のユーザー・フックおよびそれ以降のユーザー・フックはすべて実行されません。

---

**注意：** デートトラック操作がある特定のレコードに関して実行された場合には、1 つの行ハンドラ・ユーザー・フックのみが実行されます。たとえば、デートトラック・モード「UPDATE」を使用して個人レコードを更新する場合は、「更新後」ユーザー・フックのみが PER_ALL_PEOPLE_F 行ハンドラで実行されます。

---

公開済 API は、人事管理システム内でビジネス・イベントを実行するので、ビジネス・プロセスとしても知られています。

## ユーザー・フックで使用可能なデータ値

一般に、ある値が API 内で認識されている場合、その値はカスタム・ユーザー・フック・コードに使用できます。

値はすべて読込み専用です。いずれの値もユーザー・フック・ロジックによって変更することはできません。

次に示す AOL WHO 値は、ユーザー・フックでは一切使用できません。

- LAST_UPDATE_DATE
- LAST_UPDATED_BY
- LAST_UPDATE_LOGIN
- CREATED_BY
- CREATION_DATE

p_validate パラメータ値は、いずれのユーザー・フックでも使用できません。追加処理はすべて、p_validate 値に関係なく実行する必要があります。

個別の PL/SQL プロシージャ・パラメータを使用すれば、データ値をユーザー・フック・ロジックに使用できます。たいていの場合、パラメータ名称は対応するデータベースの列の名前に *p_* を接頭辞として付けたものと同じになります。たとえば、PER_ALL_PEOPLE_F 表の NATIONALITY 列は、p_nationality という対応するユーザー・フック・パラメータ名称を持ちます。

### 処理前および処理後ユーザー・フックのデータ値

- 各公開済 API の IN パラメータ値は、「処理前」と「処理後」の両方のユーザー・フックで使用できます。「処理前」フックでは、値の妥当性検査は一切行われません。
- 公開済 API の OUT パラメータ値は、「処理後」ユーザー・フックからしか使用できません。これらのパラメータ値は、コア製品ロジックを使用して導出されたものではないため、「処理前」ユーザー・フックからは使用できません。
- 公開済 API の IN OUT パラメータ値は、「処理前」と「処理後」の両方のユーザー・フックで使用できます。潜在的に無効な IN 値は、「処理前」ユーザー・フックで使用できます。公開済 API から渡された値は、「処理後」ユーザー・フックで使用できます。

行ハンドラ「挿入後」ユーザー・フックからは、挿入が可能な、または挿入中に導出される列値のみしか使用できません。

「更新後」ユーザー・フックからは 2 組の値を使用できます。新規数値および前数値です。つまり、更新されたレコードに対応する値と DML 文が実行される前にそのレコードに存在してい値です。新規数値のパラメータ名称は、データベースの列の名前に *p_* を接頭辞として付けたものと同じになります。前数値のパラメータ名称は、データベースの列の名前に *p_* を接頭辞として、*_o* を接尾辞としてそれぞれ付けたものと同じになります。たとえば、PER_ALL_PEOPLE_F 表の NATIONALITY 列の新規数値のパラメータ名称は、p_nationality になります。前数値のパラメータ名称は p_nationality_o になります。

主キー ID は別として、データベースの列が更新できない場合には、新規数値のパラメータは使用できません。接尾辞 *_o* のない対応するパラメータが依然として存在します。たとえば、PER_ALL_PEOPLE_F 表の BUSINESS_GROUP_ID 列は更新できません。「更新後」ユーザー・フックでは、p_business_group_id_o パラメータを使用できます。ただし、新規数値の p_business_group_id パラメータはありません。

「削除後」ユーザー・フックからは、*_o* が接尾辞として付けられたパラメータ名称を持つ以前の数値のみしか使用できません。接尾辞 *_o* のないパラメータを持つ主キー ID 値は使用可能です。

前数値は、行ハンドラの「更新後」と「削除後」の 2 種類のユーザー・フックでしか使用可能にできません。前数値は、「処理前」、「処理後」、「挿入後」ユーザー・フックのいずれからも使用できません。

データベースの列名が使用される場合には、30 文字という PL/SQL パラメータ名称の制限に適合するように列名の終わりが切り捨てられることがあります。

デートトラック表の列ハンドラに関しては、データ値が「挿入後」、「更新後」または「削除後」ユーザー・フックから使用可能にされた場合には必ず、システム定義の新規数値と前数値がその操作の有効日から適用されます。過去または先日付の値が必要とされる場合には、カスタム・ロジックでデータベース表からそれらの値を明示的に選択する必要があります。effective_start_date および effective_end_date 列とデートトラック・モード値が使用可能にされます。

使用可能なユーザー・フックとシステム定義のデータ値の全一覧を検索するには、PL/SQL スクリプトを実行します。

**関連項目：** API ユーザー・フック・サポート・スクリプト : 12-54 ページ

## 個別のパッケージ・プロシージャへの追加ロジックのインプリメント

ユーザー・フックを使用して API とリンクする追加ロジックは、PL/SQL サーバー側パッケージ・プロシージャ内でインプリメントする必要があります。

---

**注意：** これらのプロシージャでは、「コミット」および完全な「ロールバック」は別として、PL/SQL でインプリメントできるものはすべて実行可能です。

---

API は、ビジネス・プロセスに関連付けられている作業を完全に実行するように設計されています。すべてのデータベース変更を完了できない場合には、API は失敗に終わり、すべての変更がロールバックされます。これは、API 内で値をデータベースにコミットしないことにより達成されます。後の処理でエラーが発生した場合には、その時点まで行われたデータベース変更がすべて、自動的にロールバックされます。

**注意：** コミットや完全なロールバックはこうした機能の妨げとなるため、API コードでは一切禁止されています。これには、ユーザー・フックや追加ロジックが含まれます。コミットまたは完全なロールバック文をコミットしようとすると、ユーザー・フック機能はそれを検出し、独自のエラーを発生させます。

追加検証によって無効な値が検出された場合には、PL/SQL 例外を使用してエラーを発生させてください。これによって、現在の公開済 API のコールによって実行されたデータベース変更がすべて、自動的にロールバックされます。こうしたロールバックには、それ以前のユーザー・フックによって行われた変更がすべて含まれます。

ユーザー・フック・コードは、カスタム・コードの実行時期を決定するための決定ロジックやオプション・ロジックは一切サポートしていません。追加ロジックをユーザー・フックにリンクした場合には、API 処理ポイントに到達すると必ず、追加ロジックがコールされます。条件ロジックは、カスタム・パッケージ・プロシージャ内にインプリメントしてください。たとえば、「管理者」が 1 回の変更で 1 等級ステップのみ昇格されていることを確認する必要があるとします。役職タイプに関係なく、すべてのアサイメントに対して追加ロジックがコールされるため、等級詳細をチェックする前に「管理者」の役職をチェックする必要があるかどうかを決定してください。

### 制限

カスタム PL/SQL コードとしての追加ロジックのインプリメントには、いくつかの制限があります。サーバー側パッケージ・プロシージャのコールのみがサポートされています。ただし、同じユーザー・フックから 2 つ以上のパッケージ・プロシージャを実行することができます。カスタム PL/SQL は、次にインプリメントされている場合には、ユーザー・フックから実行することはできません。

- スタンドアロン・プロシージャ（パッケージ内に定義されていない）
- パッケージ・ファンクション
- スタンドアロン・パッケージ・ファンクション（パッケージ内に定義されていない）
- オーバーロード型バージョンが含まれたパッケージ・プロシージャ

**注意：** コミットや完全なロールバック文をカスタム PL/SQL にインプリメントすることは避けてください。API 処理の妨げとなるため、エラーが発生します。

パラメータ名称を定義するときには、実行場所のユーザー・フックで使用可能なデータ値パラメータの名称とまったく同じように定義する必要があります。パラメータは、ユーザー・フックのデータ値と同じデータ型を持っている必要があります。ユーザー・フックでは通常の暗黙 PL/SQL データ変換はサポートされていません。パッケージ・プロシージャ・パラメータはすべて、デフォルト値を一切持たない IN パラメータとして定義してください。OUT パラメータと IN OUT パラメータは、カスタム・パッケージ・プロシージャではサポートされていません。

すべてのユーザー・フックで、多数のデータ値の使用が可能です。カスタム・パッケージ・プロシージャをインプリメントするときに、すべてのデータ値をリストする必要はありません。リストする必要があるのは、カスタム PL/SQL に必要なパラメータのデータ値のみです。

使用可能なユーザー・フック、システム定義のデータ値とそれらのデータ型の全一覧を検索するには、PL/SQL スクリプトを実行します。

**関連項目：** API ユーザー・フック・サポート・スクリプト : 12-54 ページ

カスタム PL/SQL パッケージが完成したら、パッケージ作成スクリプトをデータベース上で実行して、パッケージ・プロシージャがコンパイルされるかどうかテストします。さらに、パッケージ・プロシージャが目的とする検証をテスト・データベース上で実行するかどうかテストしてください。

### 例

ある企業では、全既婚女性について、システムに入力するときに旧姓を必要とします。こうした要件はコア製品にはインプリメントされていませんが、インプリメンテーション・チームがこうした追加検証を個別のパッケージ・プロシージャの中でコード化し、API ユーザー・フックを使用してコールすることが可能です。婚姻区分が「既婚」で、性別が「女」である場合は、PL/SQL 例外を使用して、旧姓が NULL の場合にエラーを発生させます。次のサンプル・コードは、こうした検証ルールを実行するためのサーバー側パッケージ・プロシージャを示したものです。

```
Create Or Replace Package cus_extra_person_rules as
procedure extra_name_checks
  (p_previous_last_name               in varchar2
  ,p_sex                              in varchar2
  ,p_marital_status                   in varchar2
  );
end cus_extra_person_rules;
/
exit;
Create Or Replace Package Body cus_extra_person_rules as
procedure extra_name_checks
  (p_previous_last_name               in varchar2
  ,p_sex                              in varchar2
  ,p_marital_status                   in varchar2
```

```
  ) is
begin
   -- When the person is a married female raise an
   -- error if the previous last name has not been
   -- entered
   if p_marital_status = 'M' and p_sex = 'F' then
     if p_previous_last_name is null then
       dbms_standard.raise_application_error
         (num => -20999
         ,msg => 'Previous last name must be entered for married females'
         );
     end if;
   end if;
end extra_name_checks;
end cus_extra_person_rules;
/
exit;
```

## カスタム・プロシージャのユーザー・フックへのリンク

目的のデータベース上でパッケージ作成スクリプトを実行したら、カスタム・パッケージ・プロシージャを該当する API ユーザー・フックにリンクしてください。ユーザー・フックとカスタム・パッケージ・プロシージャの間のリンクは、HR_API_HOOK_CALLS 表にデータとして定義されます。

次に示すように、この表にデータを保守するための特別な API が 3 つあります。

- hr_api_hook_call_api.create_api_hook_call
- hr_api_hook_call_api.update_api_hook_call
- hr_api_hook_call_api.delete_api_hook_call

### HR_API_HOOK_CALLS

- HR_API_HOOK_CALLS 表には、特定のユーザー・フックにリンクしている各パッケージ・プロシージャに関する行が 1 行ずつ含まれている必要があります。
- API_HOOK_CALL_ID 列は一意の識別子です。
- API_HOOK_ID 列は、パッケージ・プロシージャにリンクするユーザー・フックを指定します。

  これは、HR_API_HOOKS 表に対する外部キーです。現在は、ユーザー・フック機能はパッケージ・プロシージャのコールのみサポートするので、API_HOOK_CALL_TYPE 列を 'PP' に設定する必要があります。
- ENABLED_FLAG 列は、ユーザー・フック・コールを含める必要があるかどうかを示します。

  Yes の場合は 'Y' に、No の場合は 'N' に設定してください。

- SEQUENCE 列は、フック・コールの順序を示すのに用いられます。最低値からまず処理されます。

  ユーザー・フック機能は、Oracle がアプリケーション、国別仕様および垂直市場それぞれに固有の PL/SQL を提供するためにも使用されています。1000 から 1999 までの順序番号は、Oracle 内部使用のために確保されています。

  カスタム・ロジックには、1000 未満または 2000 以上の順序番号を使用してください。可能であれば、2000 より大きい順序番号を使用することをお薦めします。Oracle 固有のユーザー・フック・ロジックがまず実行されます。したがって、Oracle の追加ロジックをカスタム・ロジック内に複製する必要はありません。

この他にも API ユーザー・フック機能によって使用されるデータが含まれている表が 2 つあります。HR_API_MODULES と HR_API_HOOKS です。

### HR_API_MODULES

HR_API_MODULES には、ユーザー・フックが含まれているすべての API コード・モジュールに関する行が 1 行ずつ含まれています。

| HR_API_MODULES メイン列 | 説明 |
|---|---|
| API_MODULE_ID | 一意の識別子 |
| API_MODULE_TYPE | API コード・モジュールのタイプを表すコード値<br>ビジネス・プロセス API（公開済 API）の場合 'BP'<br>内部行ハンドラ・コード・モジュールの場合 'RH' |
| MODULE_NAME | 値はモジュール・タイプによる<br>'BP' の場合は、CREATE_EMPLOYEE などの公開済 API の名前<br>'RH' の場合は、PER_PERIODS_OF_SERVICE などの表の名前 |

### HR_API_HOOKS

HR_API_HOOKS 表は、HR_API_MODULES 表の子表です。HR_API_HOOKS 表には、特定の API コード・モジュール内の各ユーザー・フックに関するレコードが 1 つずつ含まれています。

| HR_API_HOOKS メイン列 | 説明 |
|---|---|
| API_HOOK_ID | 一意の識別子 |
| API_MODULE_ID | 外部キー。HR_API_MODULES 表に対する親 ID |
| API_HOOK_TYPE | ユーザー・フックのタイプを表すコード値 |

API_HOOK_TYPE コードはユーザー・フックのタイプを表します。

| ユーザー・フック・タイプ | API_HOOK_TYPE |
|---|---|
| 挿入後 | AI |
| 更新後 | AU |
| 削除後 | AD |
| 処理前 | BP |
| 処理後 | AP |

**警告：** HR_API_MODULES 表や HR_API_HOOKS 表内のデータは、オラクル社によって提供および所有されるものです。HR_API_HOOK_CALLS 表内の一部のデータもまた、オラクル社により提供されています。これらの表内のデータは変更しないでください。これらの表になんらかの変更を加えた場合には、製品の性能に影響が及ぶ可能性があるだけでなく、オラクル社とのサポート契約が無効になる恐れがあります。

**注意：** これらの表内のデータは複数のソースから集められているため、データベースによって API_MODUL_ID や API_HOOK_ID の値が異なる場合があります。スクリプトを記述する場合は、こうした差異を考慮に入れてください。

**例**

たとえば、旧姓が入力されていることを確める場合には、新規個人がシステムに入力される度に、追加検証が実行される必要があります。こうした検証は、PER_ALL_PEOPLE_F 行ハンドラ「挿入後」ユーザー・フックから実行するのが最適です。

下記の PL/SQL コードは、「*create_api_hook_call*」API をコールするためのサンプル・スクリプトです。このコードは、*cus_extra_person_rules.extra_name_checks* パッケージ・プロシージャを PER_ALL_PEOPLE_F 行ハンドラ「挿入後」ユーザー・フックから実行するように、ユーザー・フック機能に知らせます。

```
declare
  --
  -- Declare cursor statements
  --
  cursor cur_api_hook is
    select ahk.api_hook_id
      from hr_api_hooks ahk
         , hr_api_modules ahm
     where ahm.module_name   = 'PER_ALL_PEOPLE_F'
       and ahm.api_module_type   = 'RH'
       and ahk.api_hook_type = 'AI'
       and ahk.api_module_id = ahm.api_module_id;
  --
  -- Declare local variables
  --
  l_api_hook_id            number;
  l_api_hook_call_id       number;
  l_object_version_number  number;
begin
  --
  -- Obtain the ID if the PER_ALL_PEOPLE_F
  -- row handler After Insert API user hook.
  --
  open cursor csr_api_hook;
  fetch csr_api_hook into l_api_hook_id;
  if csr_api_hook %notfound then
    close csr_api_hook;
    dbms_standard.raise_application_error
         (num => -20999
         ,msg => 'The ID of the API user hook was not found'
         );
  end if;
  close csr_api_hook;
  --
  -- Tell the API user hook mechanism to call the
  -- cus_extra_person_rules.extra_name_checks
  -- package procedure from the PER_ALL_PEOPLE_F row
```

```
  -- handler module 'After Insert' user hook.
  --
  hr_api_hook_call_api.create_api_hook_call
    (p_validate              => false
    ,p_effective_date        =>
                  to_date('01-01-1997', 'DD-MM-YYYY')
    ,p_api_hook_id           => l_api_hook_id
    ,p_api_hook_call_type    => 'PP'
    ,p_sequence              => 3000
    ,p_enabled_flag          => 'Y'
    ,p_call_package          =>
                  'CUS_EXTRA_PERSON_RULES'
    ,p_call_procedure        => 'EXTRA_NAME_CHECKS'
    ,p_api_hook_call_id      => l_api_hook_call_id
    ,p_object_version_number =>
                  l_object_version_number
    );
  commit;
end;
```

この例では、previous_last_name、sex、marital_status の値の更新が可能です。marital_status の変更時に同様のチェックを実行する場合には、同様の検証を PER_ALL_PEOPLE_F「更新後」ユーザー・フックから実行する必要があります。同じデータ値がこのユーザー・フックに対しても使用できるため、同じカスタム・パッケージ・プロシージャの使用が可能です。もう一度「*create_api_hook_call*」API をコールして、別の API フック・コール側定義を HR_API_HOOK_CALLS 内に作成してください。今回は、*p_api_hook_id* パラメータを PER_ALL_PEOPLE_F「更新後」ユーザー・フックの ID に設定する必要があります。

## API ユーザー・フック・プリプロセッサ・プログラム

HR_API_HOOK_CALLS 表に行を追加しても、追加ロジックがユーザー・フックから自動的にコールされるわけではありません。定義とカスタム・パッケージ・プロシージャの両方がデータベース内に作成されたら、API ユーザー・フック・プリプロセッサ・プログラムを実行してください。そうすることにより、HR_API_HOOK_CALLS 表内のコール側定義やカスタム・サーバー側パッケージ・プロシージャにリストされているパラメータをチェックすることができます。

> **注意：** 別のパッケージ本体がデータベース内に動的に構築されます。これは、フック・パッケージ本体として知られています。

フック・パッケージ本体の作成スクリプトが含まれているオペレーティング・システム・ファイルは 1 つもありません。こうしたファイルは、API ユーザー・フック・プリプロセッサ・プログラムによって動的に作成されます。各種妥当性チェックが成功したと仮定した場合、このパッケージにはハードコードされるカスタム・パッケージ・プロシージャに対するコールが含まれます。

追加ロジックが 1 つもインプリメントされない場合には、依然として対応するフック・パッケージ本体が動的に作成されます。対応するフック・パッケージ本体には、他のパッケージ・プロシージャに対するコールは一切ありません。

プリプロセッサ・プログラムは、いくつかのサーバー側 Oracle インストール・スクリプトやアップグレード・スクリプトの終わりに自動的に実行されます。このため、複数のバージョンのフック・パッケージ本体がデータベース内に存在するようになります。API ユーザー・フックを使用しない場合には、他の設定ステップは一切必要ありません。

ユーザー・フック機能は、Oracle が垂直バージョンの製品や、いくつかのアプリケーションおよび国別仕様に追加ロジックを提供する場合に使用されます。この PL/SQL に対するコールは、フック・パッケージ本体にも生成されます。

**警告：** 生成されたフック・パッケージ本体を直接編集しないことが重要です。なんらかの変更を加えた場合には、製品の性能に影響が及ぶ可能性があるだけでなく、オラクル社とのサポート契約が無効になる恐れがあります。

代替バージョンを作成することを選択した場合、代替バージョンは次にプリプロセッサ・プログラムを実行したときに失われます。こうした状況は、Oracle インストール・スクリプトまたはアップグレード・スクリプトが実行されると発生します。インプリメンテーション・チームの他の開発者がプリプロセッサ・プログラムを実行することもできます。

なんらかの変更が必要な場合には、カスタム・パッケージまたは HR_API_HOOK_CALLS 表内のコール側定義データを修正してください。次に、プリプロセッサ・プログラムを実行して、新バージョンのフック・パッケージ本体を生成します。たとえば、特定のカスタム・パッケージ・プロシージャのコールを停止する場合には、次の手順を実行します。

1. 「*hr_api_hook_call_api.update_api_hook_call*」API をコールして、*p_enabled_flag* パラメータを 'N' に設定します。
2. 最新定義が再び読み込まれ、フック・パッケージ本体が再び動的に作成されるように、API ユーザー・フック・プリプロセッサ・プログラムを実行します。

このコールをもう一度含めたい場合は、これらのステップを繰り返して、「*hr_api_hook_call_api.update_api_hook_call*」API の *p_enabled_flag* パラメータを 'Y' に設定してください。

カスタム・コールをユーザー・フックから永続的に削除する場合は、対応するコール側定義を削除します。「*hr_api_hook_call_api.delete_api_hook_call*」API をコールします。

ユーザー・フック・パッケージ本体からの実際のコールは、プリプロセッサ・プログラムが再実行されたときにのみ削除されるということを覚えておいてください。

### プリプロセッサ・プログラムの実行

プリプロセッサ・プログラムは、次の 2 通りの方法で実行できます。

- SQL*Plus の *hrahkall.sql* スクリプトを実行します。

  これによって、各種 API コード・モジュールすべてに対して、フック・パッケージ本体が作成されます。

- SQL*Plus の *hrahkone.sql* スクリプトを実行します。

  これによって、たった 1 つの API コード・モジュール（1 つのメイン API または 1 つの内部行ハンドラ・モジュール）に対して、フック・パッケージ本体が作成されます。

  このスクリプトで *api_module_id* を指定してください。必要な ID 値は、HR_API_MODULES 表内にあります。

*hrahkall.sql* スクリプトと *hrahkone.sql* スクリプトは両方とも、オペレーティング・システムの $PER_TOP/admin/sql ディレクトリ内に保存されています。

### 例

前の例を続けます。コール側定義とカスタム・パッケージ・プロシージャがデータベース内に作成されたら、次の SQL 文を使用して api_module_id を見つけ出すことができます。

```
select api_module_id
  from hr_api_modules
 where api_module_type = 'RH'
       and module_name = 'PER_ALL_PEOPLE_F';
```

次に、*hrahkone.sql* スクリプトを実行します。プロンプトが表示されたら、前述の SQL 文によって返された *api_module_id* を入力してください。そうすることにより、PER_ALL_PEOPLE_F 行ハンドラ・モジュールのユーザー・フック「挿入後」、「更新後」および「削除後」すべてに対して、フック・パッケージ本体が生成されます。

### ログ・レポート

どちらのプリプロセッサ・プログラムもログ・レポートを作成します。hrahkall.sql スクリプトはエラーのみをリストします。'Created on' 文の後にテキストがない場合には、PL/SQL エラーやアプリケーション・エラーなしに、すべてのフック・パッケージ本体が作成されています。hrahkone.sql スクリプトは終了コメントまたはエラー詳細を出力します。エラーが発生した場合は、いずれのスクリプトの終了時にも PL/SQL 例外が引き起こされます。これにより、コール側プログラムに問題の発生が通知されます。

エラーが発生した場合でも、フック・パッケージ本体コードは有効な PL/SQL を使用して作成することができます。たとえば、カスタム・パッケージ・プロシージャに使用不可能なパラメータがリストされていたとしても、フック・パッケージ本体は正常に作成されます。その特定のカスタム・パッケージ・プロシージャを実行するためのコードは作成されません。他のカスタム・パッケージ・プロシージャを同じユーザー・フックから実行する必要がある場合には、標準的な PL/SQL チェックや検証チェックのすべてに合格すれば、それらのコールを実行するためのコードも作成されることになります。

**注意：** これらのログ・レポートをチェックして、スクリプトの結果を確認することが重要です。コールが作成できなかった場合には、HR_API_HOOK_CALLS 表の対応する行も更新されます。無効コールの場合は STATUS 列が 'I' に設定され、ENCODED_ERROR 列には AOL アプリケーション・エラー・メッセージがコード化形式で挿入されます。

コード化形式は、次の PL/SQL を使用して翻訳テキストに変換できます。

```
declare
  l_encoded_error varchar2(2000);
  l_user_read_text varchar2(2000);
begin
  -- Substitute ??? with the value held in the
  -- HR_API_HOOK_CALLS.ENCODED_ERROR column.
  l_encoded_error := ???;
  fnd_message.set_encoded(encoded_error);
  l_user_read_text := fnd_message.get;
end;
```

ログ・レポートに記録された問題については、ユーザーが責任を持ってレビューおよび解決する必要があります。次のようなオプションがあります。

- カスタム・パッケージ・プロシージャのパラメータを変更する。
- 必要であれば、HR_API_HOOK_CALLS 表に定義されているデータを変更する。

問題を解決したら、プリプロセッサ・プログラムを再実行してください。

生成されるユーザー・フック・パッケージ本体の大きさは、32K 未満にする必要があります。この制限は、PL/SQL の制限です。この限界に達した場合には、各ユーザー・フックからコールされる個別パッケージ・プロシージャの数を減らす必要があります。カスタム・ロジックを結合して、プロシージャの数を減らしてみてください。

**注意：** リンクしたカスタム・パッケージ・プロシージャのサイズは 32K 以上にすることができます。32K 未満とする必要があるのは、データベース内に動的に作成されるユーザー・フック・パッケージ本体のみです。

API ユーザー・フック・アプローチを採用することの利点の 1 つは、API がコールされるたびに、ユーザー定義の追加ロジックがコールされるということです。これには、API をコールすることによってプロセス・ロジックを実行する人事管理システム・フォームまたは Web ページがすべて含まれます。

**注意：** カスタム・ロジックをコールするユーザー・フック機能は、標準製品の一部としてサポートされています。ただし、ユーザー独自のカスタム PL/SQL プロシージャのロジックについては、Oracle サポートのサポート対象外です。

## 異なるタイプのユーザー・フックを使用する場合の推奨事項

次の 2 つのカテゴリの検証ルールについて考えてみましょう。

- データ項目ルール

  表内のあるフォームまたは列のある特定のフィールドと関連付けられたルール。たとえば、割り当てられた等級は、割り当てられた役職に対して常に有効なものである必要があります。

- ビジネス・プロセス・ルール

  特定のトランザクションまたはプロセスと関連付けられたルール。たとえば、サブ・アサイメントを作成するときには、特別の付加セグメント値を含める必要があります。

### データ項目ルール

公開済 API は、ビジネス・プロセスをサポートするように設計されています。つまり、個々のデータ項目を複数の API によって変更することが可能です。特定のデータ項目（表列）に対して追加のデータ検証を実行するには、内部行ハンドラ・モジュール・ユーザー・フックを使用する必要があります。

内部行ハンドラ・コード・ユーザー・フックから追加ロジックをインプリメントすることによって、その列値が変化する可能性のあるケースをすべて含めることができます。そうでなければ、そのデータベースの列を設定または変更することのできる API をすべて識別する必要があります。

データの妥当性検査には、「挿入後」、「更新後」または「削除後」ユーザー・フックを使用してください。これらのユーザー・フックは、そのデータベースの表の行と関連付けられている検証全体が完了しなければ実行されないので最適です。カスタム・ロジックに渡されるデータ値はすべて、コア製品に関する限り有効です。

フック・コール定義が 1999 より大きな順序番号で作成されれば、Oracle 国別仕様または垂直市場固有のロジックも正常に実行されているはずです。

**注意：** 追加検証を「挿入後」ユーザー・フックにインプリメントした場合、関連するデータ値が更新可能であれば、「更新後」ユーザー・フックから同様のロジックを除外することを検討する必要があります。

前数値（DML の前）は、「更新後」および「削除後」ユーザー・フックから使用可能です。

### ビジネス・プロセス・ルール

発生した特定のビジネス・イベントを検出する場合や、特定の公開済 API の特定の追加ロジックのみを実行する場合には、「処理前」および「処理後」ユーザー・フックを使用してください。

その API 全体のコア製品検証はすべて完了しているはずなので、可能な限り「処理後」ユーザー・フックを使用してください。「処理前」ユーザー・フックを使用する場合には、すべてのデータ値がカスタム・ロジックでは無効である可能性があるということを考慮に入れる必要があります。その時点では、コア製品検証は少しも実行されていません。HR_LOOKUPS ビューへの参照、HR_LOOKUPS に結合されるすべてのビュー、および参照コード検証は「処理前」ユーザー・フックで実行できません。参照コード検証に影響を与える値は、このポイントの後まで導出および設定されません。

「処理前」および「処理後」ユーザー・フックで提供されるデータ値は、API に渡される値と同じになります。更新型のビジネス・プロセスに関しては、API のコール側が必須パラメータと実際に変更する値を指定してください。API のコール側がパラメータ値を明示的に指定しない場合には、次のシステムに保存されているデフォルト値が使用されます。

```
データ型    デフォルト値
---------  -----------------
varchar2   hr_api.g_varchar2
number     hr_api.g_number
date       hr_api.g_date
```

API のコール側が指定したパラメータに応じて、これらのデフォルト値が「処理前」および「処理後」ユーザー・フックに提供されます。つまり、データベース内の既存の列値は、API のコール側コードが偶然にも同じ新規数値を渡した場合にのみ提供されます。実際のデータベース値が必要とされる場合には、カスタム・パッケージ・プロシージャがデータベースから明示的に選択する必要があります。

これが、「更新後」および「削除後」ユーザー・フックが望ましいとされるもう 1 つの理由です。行ハンドラ・ユーザー・フックには、常に実際のデータ値が提供されます。システムのデフォルト値はすべて、行ハンドラ・モジュール内のそれらの既存のデータベース列値に再設定されているはずです。これらのユーザー・フックの追加ロジックはすべて、システムに保存されているデフォルト値と関連する必要があります。

「処理後」追加ロジックによって前のデータベース値にアクセスする必要がある場合には、別のユーザー・フックを使用してください。関連するデータベースの行がすべて修正されているはずなので、「処理後」ユーザー・フックを使用することはできません。また、前の数値は、ユーザー・フック機能では提供されません。API 固有の追加ロジックが前の数値を必要とする場合には、「前処理」ユーザー・フックでそれらの値を明示的に選択する必要があります。

### ユーザー・フックおよび代替インタフェース API

代替インタフェース API は、汎用 API の代替バージョンを提供します。現在は、国別仕様バージョンや垂直仕様バージョンの汎用 API があります。

たとえば、*create_us_employee* と *create_gb_employee* の 2 つは、汎用「*create_employee*」API への代替インタフェースです。これらの代替インタフェースは、汎用 API のパラメータへの特定の国別パラメータのマッピング方法を明確にします。

将来は、この他にもエレメントおよび入力パラメータなど、特定の汎用機能の導入をサポートする代替 API が提供される可能性があります。

**注意：** 代替インタフェース API にはユーザー・フックは提供されていません。ユーザー・フックが提供されているのは汎用 API のみです。この例では、「*create_us_employee*」や「*create_gb_employee*」API ではなく、「*create_employee*」API にユーザー・フックが提供されています。

代替インタフェース API は常に汎用 API を実行することによって処理を実行するため、汎用 API ユーザー・フックの追加ロジックはすべて、代替 API がコールされると自動的に実行されます。これによって、一貫した追加ロジックの実行が保証されるため、リンクの設定や保守の管理が容易になります。

#### 例 1

従業員アサイメントの役職および給与計算コンポーネントの追加検証を実行して、「マシン労働者」のみが「週次」給与計算に含まれるようにするとします。新規アサイメントの作成時もしくは既存のアサイメントの更新時に値の設定を可能にする公開済 API はいくつもあります。

提案。追加検証をカスタム・サーバー側パッケージ・プロシージャにインプリメントします。これを PER_ALL_ASSIGNMENTS_F 表の内部行ハンドラ・モジュール内にある「挿入後」と「更新後」の 2 つのユーザー・フックにリンクします。

#### 例 2

カスタム表を持っていて、新規従業員をシステム内に作成するとき、もしくは既存の応募者を従業員に変換するときに、その表の中にデータを作成するとします。カスタム表内のデータを別のシナリオの中に作成する必要は一切ありません。

提案。サード・パーティ表挿入 DML 文をカスタム・サーバー側パッケージ・プロシージャにインプリメントします。これを「*create_employee*」API モジュールの「処理後」と、「*hire_applicant*」API モジュールの「処理後」の 2 つのユーザー・フックにリンクします。

## データベース・トリガーとの比較

追加ロジックのインプリメントに関しては、ユーザー・フックにはデータベース・トリガーよりも数多くの利点があります。

- データベース・トリガーは、個々の表の DML 文に対してしか定義できません。特定のビジネス・イベントのコンテキストは、イベント詳細が表のいずれの列にも保持されていないために、表レベルでは使用できない場合があります。
- サーバー側パッケージ・プロシージャの実行に比べると、データベース・トリガーの実行は効力がありません。
- 変更表制限により、変更中の表の行からの値の選択が禁止されています。このため、データベース・トリガーでは複雑な複数行検証はインプリメントされていません。API ユーザー・フックにはこのような制限はまったくないので、こうした複雑な検証は API ユーザー・フックからインプリメントできます。
- デートトラック表には、データベース・トリガーからなんらかの有用なロジックをインプリメントすることはきわめて困難です。多くのデートトラック・モードでは、単一のトランザクションが同じデータベース表内の複数の行に影響を及ぼす可能性があります。デートトラック・レコードの日付の付いたインスタンスはそれぞれ、物理的に別のデータベース行に保持されます。

  たとえば、挿入を起動するデータベース・トリガーでは、作成中の新規レコードとデートトラック「UPDATE」操作による挿入行の相違を知ることはできません。

  > **注意：** デートトラック「UPDATE」では、1 つの *insert* 文と 1 つの *update* 文が実行されます。デートトラック・モードのコンテキストは、データベース表レベルで失われます。変更表制限により、データベース・トリガでこれを再導出することは不可能です。

- デートトラック表の行ハンドラ・ユーザー・フックを用いれば、より多くのコンテキストやデータ値の使用が可能です。「挿入後」ユーザー・フックは、新規レコードの作成時にのみ実行されます。デートトラック・モード名は、「更新後」および「削除後」ユーザー・フックで使用可能です。これらのユーザー・フックでは、レコードが変更されている日付の範囲も使用可能です。*validation_start_date* 値は、レコードが現在のデートトラック操作による影響を受ける最初の日になります。レコードが影響を受ける最後の日は、*validation_end_date* です。

### API ユーザー・フック・サポート・スクリプト

使用可能なユーザー・フックとシステム定義のデータ値の全一覧を検索するには、SQL*Plus の *hrahkpar.sql* スクリプトを実行します。このスクリプトは、オペレーティング・システムの $PER_TOP/admin/sql ディレクトリ内に保存されています。出力が長いので、出力をオペレーティング・システム・テキスト・ファイルにスプールすることをお薦めします。

ユーザー・フック・プリプロセッサ・プログラムは、2 通りの方法で実行できます。各種 API コード・モジュールすべてに対してフック・パッケージ本体を作成するには、SQL*Plus の *hrahkall.sql* スクリプトを実行します。たった 1 つの API コード・モジュール（1 つのメイン API または 1 つの内部行ハンドラ・モジュールなど）に対してフック・パッケージ本体を作成するには、SQL*Plus の *hrahkone.sql* スクリプトを実行します。この 2 番目のスクリプトで *api_module_id* を指定してください。必要な *api_module_id* 値は、HR_API_MODULES 表から入手できます。*hrahkall.sql* スクリプトと *hrahkone.sql* スクリプトは両方とも、オペレーティング・システムの $PER_TOP/admin/sql ディレクトリ内に保存されています。

## 基本ブロックとしての API の使用方法

この製品に搭載されている API コード・ファイルを独自の目的にあわせて直接編集することは絶対にしないでください。

---

**警告：** なんらかの変更を加えた場合には、製品の性能に影響が及ぶ可能性があるだけでなく、オラクル社とのサポート契約が無効になる恐れがあります。また、製品のアップグレードもできません。

---

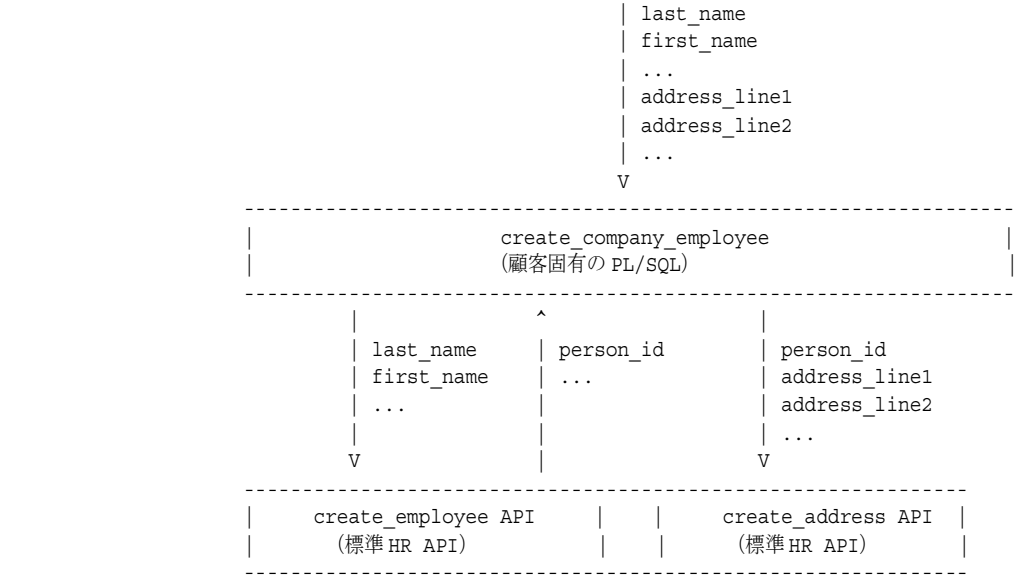
Oracle Applications では、公開済 API の直接コールがサポートされています。オラクル人事管理システム製品セットの一部として記述された他のサーバー側パッケージ・プロシージャまたは関数の直接コールは、特に明記されていない限りサポートされていません。

システム定義の API を使用してカスタム・ロジックを追加するための方法がいくつかサポートされています。API ユーザー・フック機能に加えて、公開済 API を基本ブロックとして使用することによって、カスタム API を構築することができます。

#### 例

新入社員が入社するたびに、従業員の自宅住所を入手することになっています。全従業員が住所を持っていることが前提のレポートを実行するため、住所詳細を人事システムに記録する必要があります。

独自の API を記述して、住所を持つ新規従業員を作成することも可能です。こうした API は、標準「*create_employee*」API をコールした直後に、標準「*create_address*」API をコールします。




API ユーザー・フックでは、データ値を変更することは一切できません。このため、基本ブロック・アプローチを使用することによって、公開済 API がコールされる前に、デフォルトにするか、なんらかの値を設定することができます。

基本ブロック・アプローチを使用する場合の大きな欠点は、Oracle が提供するフォームまたは Web ページがカスタム API をコールすることはないということです。ユーザー・インタフェースが必要とされる場合には、ユーザー独自のカスタム・フォームまたは Web ページを作成して、カスタム API に対するコールをインプリメントする必要があります。

## Oracle Forms におけるオブジェクト・バージョン番号の処理方法 :

API をコールする独自のフォームを記述する場合には、オブジェクト・バージョン番号を正しく管理するための追加のフォーム・ロジックをインプリメントする必要があります。これは、フォームが同じコミット単位の 2 つ以上の行を処理するためには欠かせません。

### 例

たった 1 つのフォームのブロック・アイテムのみ使用してオブジェクト・バージョン番号を保持した場合に発生する可能性のある次の例について考えてみましょう。

**1.** ユーザーは、2 つの行を問合せて、両方とも更新します。

```
     OVN in
Row  Database   OVN in Form
---  --------   -----------
A    6          6
```

```
B     3          3
```

**2.** ユーザーは、コミットをクリックします。

行 A にはユーザー・エラーがないので、その API で検証されます。データベースの OVN が更新され、新規 OVN がフォームに返されます。

```
      OVN in
Row   Database   OVN in Form
---   --------   -----------
A     7          7
B     3          3
```

**3.** フォームは、行 B に対して API を再度コールします。

今回は、ユーザー入力の変更に検証エラーがあります。フォームにエラー・メッセージが表示され、Forms はデータベースにロールバックを発行します。しかし、フォーム内の行 A の OVN は現在、データベース内の OVN と異なります。

```
      OVN in
Row   Database   OVN in Form
---   --------   -----------
A     7          6
B     3          3
```

**4.** ユーザーは、行 B に関する問題を訂正して、再度コミットします。

現在は、この API は、行 A に対する変更を検証するとエラーになります。2 つの OVN は異なります。

**解決策**

この問題を解決するには、非実表項目を使用して新規バージョン番号を保持します。この項目は、問合せ時には挿入されません。

**1.** ユーザーは、2 つの行を問合せて、両方とも更新します。

```
      OVN in                   New_OVN
Row   Database   OVN in Form   in Form
---   --------   -----------   -------
A     6          6
B     3          3
```

**2.** ユーザーは、コミットをクリックします。

行 A は有効なので、データベースの OVN が更新され、新規 OVN がフォームに返されます。

---

**注意：** フォームの実際の OVN は更新されません。

---

```
     OVN in                  New_OVN
Row  Database   OVN in Form   in Form
---  --------   -----------   -------
A    7          6             7
B    3          3
```

**3.** フォームは、行 B に対して API を再度コールします。

検証が失敗に終わり、フォームにエラー・メッセージが表示されます。Forms はデータベースにロールバックを発行します。

```
     OVN in                  New_OVN
Row  Database   OVN in Form   in Form
---  --------   -----------   -------
A    6          6             7
B    3          3
```

**4.** ユーザーは、行 B に関する問題を訂正して、再度コミットします。

API がコールされて、行 A がもう一度検証されます。OVN 値は渡されますが、NEW_OVN は渡されません。現在はデータベース内の OVN と渡された OVN が一致しているため、エラーはありません。API は更新済 OVN 値を戻します。

```
     OVN in                  New_OVN
Row  Database   OVN in Form   in Form
---  --------   -----------   -------
A    7          6             7
B    3          3
```

**5.** 行 B を検証するために、API がもう一度コールされます。

検証が完了すると、データベースの OVN が更新され、新規 OVN 値がフォームに返されます。フォームとデータベースへのコミットが完了しました。

```
     OVN in                  New_OVN
Row  Database   OVN in Form   in Form
---  --------   -----------   -------
A    7          6             7
B    4          3             4
```

ユーザーが再問合せをせずに同じ行を再度更新した場合には、どのような事態が発生するでしょうか？前のステップから続行してください。

**6.** ユーザーが行 A の更新を開始すると、ロック時トリガーが起動します。

このトリガーは、New_OVN が NULL でない場合は OVN を更新します。（理論上は、ロック時トリガーは前のコミットが成功した場合にのみ起動します。このため、New_OVN はデータベースの OVN 値になります）。

```
     OVN in                  New_OVN
```

```
Row   Database   OVN in Form   in Form
---   --------   -----------   -------
A     7          7             7
```

**7.** 次に、ロック時トリガーは、OVN を使用してロックを取り出す API をコールします。

OVN 値が一致しているので、ロックは成功に終わります。

```
      OVN in                    New_OVN
Row   Database   OVN in Form    in Form
---   --------   -----------    -------
A     7          7              7
```

**8.** ユーザーは更新を続行します。更新 API がコールされ、コミットが成功に終わります。

```
      OVN in                    New_OVN
Row   Database   OVN in Form    in Form
A     8          7              8
```

ユーザーが更新ではなく、削除を実行した場合にも、on_lock は同様の働きをします。Key_delrec がクリックされると、削除 API がコールされ、p_validate が TRUE に設定されます。そうすれば、行をデータベースから削除しなくても、削除は有効になります。

このため、New_OVN が NULL でない場合には、フォーム内の OVN 値を New_OVN に設定する必要があります。削除ロジックは、データベースの中で OVN 値で呼ばれるようにします。

しかし、考慮する必要のある特別な状況があります。ユーザーは行を更新することができます（新規 OVN 値が API から返されます）。同じコミット単位の次の行の更新は失敗に終わります。ユーザーは最初の行に戻り、その行を削除することを決定します。record_status が問合せであれば、key_delrec でコピーを実行するだけで、New_OVN のフォームの OVN へのコピーを停止できます。

### サンプル・コード（等級レート値を使用）

次に示す例では、前述の内容が用いられています。この例では、<block_name>.object_version_number が実表項目で、
<block_name>.new_object_version_number が非実表項目です。

### ON_INSERT トリガーによってコールされるフォーム・プロシージャ

```
procedure insert_row is
begin
  --
  -- Call the api insert routine
  --
  hr_grade_api.create_grade_rate_value
    (<parameters>
    ,p_object_version_number => :<block_name>.object_version_number
```

```
    ,p_validate              => false
    );
end insert_row;
```

### ON_UPDATE トリガーによってコールされるフォーム・プロシージャ

```
procedure update_row is
  l_api_ovn number;
begin
  -- Send the old object version number to the API
  l_api_ovn := :<block_name>.object_version_number;
  --
  -- Call the api update routine
  --
  hr_grade_api.update_grade_rate_values
    (<parameters>
,p_object_version_number => l_api_ovn
    ,p_validate           => false
    );
  -- Remember the new object version number returned from the API
  :<block_name>.new_object_version_number := l_api_ovn;
end update_row;
```

### ON_DELETE トリガーによってコールされるフォーム・プロシージャ

```
procedure delete_row is
begin
  --
  -- Call the api delete routine
  --
  hr_grade_api.delete_grade_rate_values
    (<parameters>
    ,p_object_version_number => :<block_name>.object_version_number
    ,p_validate           => false
    );
end delete_row;
```

### KEY_DELREC トリガーによってコールされるフォーム・プロシージャ

```
procedure key_delrec_row is
  l_api_ovn number;
  l_rec_status varchar2(30);
begin
  -- Ask user to confirm they really want to delete this row.
  --
  -- Only perform the delete checks if the
  -- row really exists in the database.
  --
```

```
l_rec_status := :system.record_status;
if (l_rec_status = `QUERY') or (l_rec_status = `CHANGED') then
  --
  -- If this row just updated then the
  -- new_object_version_number will be not null.
  -- If that commit was successful then the
  -- record_status will be QUERY, therefore use
  -- the new_object_version_number. If the commit
  -- was not successful then the user must have
  -- updated the row and then decided to delete
  -- it instead. Therefore just use the
  -- object_version_number.
  --(Cannot just copy the new_ovn into ovn
  -- because if the new_ovn does not match the
  -- value in the database the error message will
  -- be displayed twice. Once from key-delrec and
  -- again when the on-lock trigger fires.)
  --
  if (:<block_name>.new_object_version_number is not null)  and
(l_rec_status = 'QUERY') then
    l_api_ovn := :<block_name>.new_object_version_number;
else
l_api_ovn := :<block_name>.object_version_number;
end if;
  --
  -- Call the api delete routine in validate mode
  --
hr_grade_api.delete_grade_rate_values
  (p_validate                => true
    ,<parameters>
    ,p_object_version_number => l_api_ovn
    ,p_validate              => true
    );
end if;
--
delete_record;
end key_delrec_row;
```

### ON_LOCK トリガーによってコールされるフォーム・プロシージャ

```
procedure lock_row is
  l_counter number;
begin
  l_counter := 0;
  LOOP
    BEGIN
    l_counter := l_counter + 1;
    --
```

```
    -- If this row has just been updated then
    -- the new_object_version_number will be not null.
    -- That commit unit must have been successful for the
    -- on_lock trigger to fire again, so use the
    -- new_object_version_number.
    --
    if :<block_name>.new_object_version_number is not null then
       :<block_name>.object_version_number :=
           :<block_name>.new_object_version_number;
    end if;
    --
    -- Call the table handler api lock routine
    --
    pay_grr_shd.lck
      (<parameters>
      ,p_object_version_number => :<block_name>.object_version_number
       );
    return;
  EXCEPTION
    When APP_EXCEPTIONS.RECORD_LOCK_EXCEPTION then
      APP_EXCEPTION.Record_Lock_Error(l_counter);
    END;
  end LOOP;
end lock_row;
```

# 13

# Oracle HRMS Data Pump

## Oracle HRMS Data Pump

この章では、Oracle HRMS Data Pump を理解して使用するために必要な情報を提供します。この情報を理解するためには、次の事項を含めて、オラクル人事管理システム製品アーキテクチャの機能上および技術上の知識が必要となります。

- オラクル人事管理システムのデータ・モデル、およびデートトラックの重要性。
- API 計画、および API を直接コールする方法。
- PL/SQL のコーディング方法。Data Pump で従来データを使用するためには、通常は PL/SQL コードによるデータ変換が必要となります。
- コンカレント・プロセスの実行を制御する人事管理システムのパラメータ（たとえば、プロセスのパラレル実行のため）。

### 制限

この章は、Data Pump スキーマのすべてを詳細に説明するものではありません。いくつかの表に関しては必要に応じて詳しい説明が追加されています。多くの場合、PL/SQL ルーチンを使用してこれらのバッチ・インタフェース表にデータを挿入します。

Oracle HRMS Data Pump は、オラクル人事管理システムから提供されるすべての API をサポートしているわけではありません。サポートされる API のリストは『*Oracle HRMS* ユーザーズ・ガイド』を参照してください。将来のリリースではその他の API もサポートされる予定です。

Data Pump 表からデータをパージするときには、将来外部データをロードするときに必要となる可能性のあるユーザー・キーの情報を削除しないように特に注意してください。参照 : ユーザー・キー値 : 13-29 ページ

### 内容

この章は次の項から構成されています。


- 概要 : 13-2 ページ

  主なコンポーネントおよび特別機能を含めて、Data Pump の概要を説明します。

- Data Pump の使用方法 : 13-6 ページ

  高レベルで Data Pump を使用する手順を説明します。各ステップについては次の項で詳しく説明します。

  - メタマッパーの実行 : 13-8 ページ
  - バッチ表へのデータのロード : 13-14 ページ
  - Data Pump プロセスの実行 : 13-17 ページ
  - エラーの検索と修正 : 13-19 ページ
  - データのパージ : 13-23 ページ

- サンプル・コード : 13-24 ページ

  バッチ・ライン・プロシージャのコールする方法を説明します。

- 生成済インタフェースの使用方法に関する注意 : 13-27 ページ

  メタマッパー・プロセスによって各 API 用に生成された PL/SQL パッケージを使用および表示するときに考慮する必要のあるいくつかの要素を説明します。

- Data Pump で使用可能なユーティリティ・プロシージャ : 13-31 ページ

  HR_PUMP_UTILS パッケージから提供されるユーティリティ・プロシージャについて説明します。

- 表およびビューの説明 : 13-32 ページ

  Data Pump で使用する特定の表およびビューについて説明します。


## 概要

オラクル人事管理システムはビジネス・プロセスに関連する一組のシステム定義 API を提供しています。データをロードするときには、必ずこれらの API を使用することをお薦めします。システム定義の API ではシステムのすべてのビジネス・ルールが尊重されるため、システムにロードされるデータの整合性が保証されます。

Oracle HRMS Data Pump は、オラクル人事管理システム表へのバッチ・データのロードに関連する共通タスクを単純化および標準化することにより、迅速なインプリメンテーションをサポートします。これは、サポートされる API を使用して、データ・ロードのタスクを単純化する、システム定義のバッチ表および標準プロセスを提供することにより可能となります。

Oracle Data Pump を使用して次のような作業を行うことができます。

1. 外部システムからのデータ項目を適切な API のパラメータ値にマップする。

   データは API のパラメータにマップされるため、ユーザーは人事管理システム・データ・モデルの複雑さを意識する必要はありません。たとえば、従業員を作成するには、複数の表に正しくデータを挿入する必要があります。「create_employee」API は、ユーザーが渡したパラメータ値を使用してこれを自動的に実行します。

   Data Pump の特別機能により、API パラメータのシステム ID としてユーザー値を使用できます。これらの値は Data Pump によって自動的に変換されます。

2. ユーザーのデータを 1 つの汎用バッチ・ライン表にロードする（ユーザーがバッチ・ロード・プロセスを管理するための 1 つのバッチ・ヘッダー表もあります）。

   Data Pump では 1 つの汎用バッチ・ライン表が使用されます。Data Pump は API ごとに固有のビューを生成し、ユーザーが API のパラメータ名を使用して、各 API のデータを容易にレビューおよび更新できるようにします。

   また、ユーザーの外部データを汎用バッチ・ライン表に挿入するための PL/SQL インタフェース・ルーチンもあります。

3. バッチ表内の各データ / ラインに対して適切な API を自動的にコールする標準プロセスを実行する。

## Data Pump のコンポーネント

Data Pump は次のコンポーネントから構成されます。

### メタマッパー・プロセス

このプロセスは、ユーザーが使用するサポート対象の各 API モジュールについて、特定の PL/SQL プロシージャおよびビューを生成します。

メタマッパーは、バッチ表内のデータの検査および更新に使用する一組のビューを生成するために使用されます。たとえば、データを訂正したり、データのロード順序を変更することが考えられます。

---

**注意：** メタマッパーはインストール・プロセスと似ています。システム定義の API を使用してデータの検証またはロードを行う前に、メタマッパーを実行する必要があります。

---

#### バッチ・ヘッダー表とバッチ・ライン表

これら 2 つの表は、外部データからのヘッダー情報およびライン情報を保持するために使用されます。

- HR_PUMP_BATCH_HEADERS
- HR_PUMP_BATCH_LINES

---

**注意：** メタマッパーは、HRDPV_<API Procedure Name> という名称のバッチ・ライン表に基づいてビューを作成します。たとえば、HRDPV_CREATE_EMPLOYEE となります。

---

#### PL/SQL ルーチン

バッチ・ライン表に外部データまたは従来データを挿入するために、システム定義の生成済 PL/SQL ルーチンを使用します。メタマッパーは、Data Pump でサポートされる API のそれぞれに対して別個のルーチンを生成します。

- HR_PUMP_UTILS.CREATE_BATCH_HEADER(...)
- HRDPP_<API Procedure Name>.INSERT_BATCH_LINES

  たとえば、HRDPP_ CREATE_EMPLOYEE .INSERT_BATCH_LINES

特定のプロシージャについてパラメータ・オプションの詳細情報を提供するヘルプ・ルーチンもあります。

- HR_PUMP_META_MAPPER.HELP (
  <package_name>, <procedure_name>)

#### Data Pump エンジン・プロセス

Data Pump エンジン・プロセスは、データ検証およびロード操作を実際に実行する標準のコンカレント・プロセスです。2 つのパラメータが使用されます。

- バッチ名
- プロセス・モード

## Data Pump の特別機能

Data Pump で提供される特別機能は次のとおりです。

### ユーザー・キー

Data Pump では、オラクル人事管理システムにロードするレコードを一意に識別するデータ項目の組合せを定義できます。たとえば、個人のデータをロードする場合、オラクル人事管理システムでその個人を識別するために、姓、名、生年月日および性別の組合せを使用できます。

これらのユーザー・キー定義は HR_PUMP_BATCH_LINES_USER_KEYS 表に格納します。

### 実際の値の使用

ほとんどすべての場合には、オラクル人事管理システムでシステム値を確認する必要なしに、実際の名称または値を使用してデータをロードできます。名称から ID への変換はユーザーからは意識されません。たとえば、オラクル人事管理システムの JOB_ID を確認する必要なしに実際の役職名を使用できます。また、コード値が 'M' であることを知らない場合でも、性別の値として「男」を使用できます。

### バッチ・ロード・プロセスの自動パラレル処理

Data Pump は、マルチプロセッサ・システムの場合、追加コードを必要とすることなく、パラレル処理を自動的にサポートします。PAY_ACTION_PARAMETERS 表で THREADS の行を挿入または更新することにより、この処理をオンにします。

これは、オラクル人事管理システムで給与計算処理およびその他のプロセスのパラレル処理を制御するパラメータと同じものです。

---

**注意：** パラレル処理を使用する場合は、同じスレッド内で実行する必要のあるトランザクションをグループ化するために、バッチ・ラインの中で P_LINK_VALUE パラメータを使用します。

---

### ユーザーによる操作順の明示的指示

バッチ・ラインで関連データをロードするときには、いくつかの操作を厳密な順序で実行する必要があります。たとえば、従業員の給与情報は、従業員レコードが作成された後で入力する必要があります。

Data Pump では、P_USER_SEQUENCE パラメータを使用して、バッチ・ラインの処理順序を制御します。

---

**注意：** Data Pump はユーザーが入力した順序番号を検証できません。Data Pump は指示された順序をそのまま受け入れて処理しようとします。ユーザーが誤った数字を使用した場合、正しくない順序でデータのロードが行われた際に検証エラーが戻されることがあります。参照 : Data Pump プロセスの実行 : 13-17 ページ

---

### 検証モード操作

Data Pump コンカレント・プロセスを発行するときには、検証モードで実行するように選択できます。これにより、バッチおよびバッチの関連レコードの中のエラーを確認し、人事管理システム・データベースにコミットされる前に変更することができます。

### バッチの処理

Data Pump を実行すると、プロセスは処理が終了していないデータのみをロードします。つまり、バッチを実行し、ラインにエラーがあれば訂正してから、同じバッチを再実行できます。バッチの全ラインが正常にロードされるまで、このプロセスを繰り返すことができます。

これを行うためには、同じバッチ名を使用してコンカレント・プロセスを発行します。未処理のラインおよびエラーのあるラインはすべて自動的に再処理されます。

### ロギング・オプション

Data Pump には、プロセスを実行するときにエラーを検出するための数多くのロギング・オプションがあります。

## Data Pump の使用方法

Data Pump を使用する場合は、次の順序でタスクを実行します。

1. 外部データのロードに必要な、サポートされる API モジュールを選択し、メタマッパーを実行してこれらの API 用のインタフェース・プロシージャを生成します。

   **関連項目：** メタマッパーの実行 : 13-8 ページ

2. システム定義の PL/SQL ルーチンおよびメタマッパーによって作成されたプロシージャを使用して、外部データを Data Pump 表に転送します。

   **関連項目：** バッチ表へのデータのロード : 13-14 ページ

---

**注意：** ユーザー・キーを必要とする各エンティティには、一意識別子として使用する値を含める必要があります。たとえば、create_employee のパラメータ、P_PERSON_USER_KEY および P_ASSIGNMENT_USER_KEY です。

---

3. オプション。データをロードする前に、Data Pump を検証モードで実行してデータをチェックおよび訂正します。

   **関連項目：** Data Pump プロセスの実行 : 13-17 ページ

4. Data Pump を実行し、バッチ表からオラクル人事管理システム表にデータをロードします。

   ---

   **注意：** 最初にレコードをロードするときに、Data Pump はバッチ・ラインからのユーザー・キー値、および API から生成された一意キー ID を HR_PUMP_BATCH_LINE_USER_KEYS 表に自動的に挿入します。この組合せは、オラクル人事管理システム内の既存レコードを更新する、その後のデータ・ロードのすべてで使用されます。

   たとえば、P_PERSON_USER_KEY = USER_KEY_VALUE および PERSON_ID = UNIQUE_KEY_ID です。

   ---

5. エラーを確認して原因を解決します。

   **関連項目：** エラーの検索と修正 : 13-19 ページ

6. 必要な場合は、訂正したバッチ・ラインをロードするために Data Pump を再実行します。

   **関連項目：** Data Pump プロセスの再実行 : 13-22 ページ

   すべてのラインが正常にロードされるまで、5 および 6 を繰り返します。

7. オプション。バッチ表からデータをパージします。

   **関連項目：** データのパージ : 13-23 ページ

## メタマッパーの実行

システムによっては、外部データのロードにシステム定義の API をすべて使用する必要はない場合があります。選択したすべての API または 1 つの API に対して、メタマッパーを実行します。メタマッパーは、各 API 用に特定の PL/SQL パッケージおよびビューを生成します。

> **注意：** オーバーロードされたインタフェースを持つ API の場合、メタマッパーは最新のインタフェース用のコードのみを生成します。最新のインタフェースは、必須パラメータの数が一番多いインタフェースです。

すべての API 用にパッケージおよびビューを生成するには、次の SQL*PLUS コマンドを使用します。

```
sql> execute hr_pump_meta_mapper.generateall;
```

1 つの API 用にパッケージおよびビューを生成するには、次の SQL*PLUS コマンドを使用します。

```
sql> execute hr_pump_meta_mapper.generate( <package_name>,<procedure_name>);
```

例 :

```
sql> execute hr_pump_meta_mapper.generate( 'hr_employee_api', 'create_employee' );
```

ビューの命名規則は hrdpv_<api_module_name>、PL/SQL パッケージの命名規則は hrdpp_<api module name> です。名称が 30 バイトを超えないかぎりこの規則が適用されます。30 バイトを超える場合、名称は 30 バイトに切り捨てられます。この例では、ビューの名称は hrdpv_create_employee、そしてパッケージの名称は hrdpp_create_employee です。

ビューを使用して、従来データを人事管理システム・スキーマまたはバッチ表に挿入したり、バッチ・ライン表の中の既存データを更新できます。PL/SQL パッケージには、外部システムからバッチ・ライン表にデータを簡単に挿入できるようにする insert_batch_lines プロシージャと、バッチ・ライン表の中の行に対して API を実行するコール・プロシージャが含まれています。

> **注意：** Data Pump を使用してデータをロードする前にメタマッパーをコールする必要があります。必要なすべての API に対してメタマッパーをコールした後で、コンカレント・マネージャを再起動してから Data Pump を実行します。

## メタマッパーから生成されるビュー

メタマッパーは各 API について、API のパラメータを反映して HR_PUMP_BATCH_LINES 表のビューを生成します。これにより、行値の検査および更新がより容易になります。ビューの名称は API の名称を反映しています。たとえば、HRDPV_CREATE_EMPLOYEE です。このビューの全一覧は、13-32 ページの「表およびビューの説明」を参照してください。

メタマッパーは API のパラメータに加えて、常に次の列をビューに作成します。

| 列 | 説明 |
|---|---|
| BATCH_ID | HR_PUMP_BATCH_HEADERS への外部キー。 |
| BATCH_LINE_ID | HR_PUMP_BATCH_LINES への外部キー。<br>hr_pump_batch_lines_s 順序を使用して生成された主キー。 |
| API_MODULE_ID | HR_API_MODULES への外部キー。それぞれの行でどの API をコールするかを Data Pump に知らせる。 |
| LINE_STATUS | この API のロード・ステータス :<br>'U' - 未処理。すべての行の初期値はこの値にする必要があります。<br>'C' - 完了。API コールに成功し、変更がコミットされました。<br>'E' - エラー。<br>'V' - 検証済。API コールに成功したが、変更はされていません。 |
| USER_SEQUENCE | 処理順序の管理に使用。たとえば、ある従業員のレコードの作成後に、その従業員の住所が必ずロードされるようにします。 |
| LINK_VALUE | 一意の link_value を使用して、複数の行を 1 つのバッチにリンクします。パラレル処理を使用して、バッチ内の関連する行が必ず一緒に処理されるようにするときは、この値を設定します。 |

メタマッパーは、特定の API 用にその他の列も作成します。たとえば、従業員作成ビューには次のような列があります。

- P_EFFECTIVE_DATE
- P_MANAGER_FLAG
- P_ASSIGNMENT_USER_KEY

ユーザーが値を確認できるように、API から戻された PL/SQL OUT 値を反映するその他の列も作成されます。例 :

- P_NO_MANAGERS_WARNING

ユーザーは、API の特定のパラメータがバッチ・ライン表のどの列にあるかを知る必要はありません。

### 必須の列

ビューを使用してバッチ・ライン表にデータを挿入する場合、バッチ・ライン挿入プロシージャに必要なデータに加えて、次の情報も必要になりますので注意してください。

- batch_line_id

  hr_pump_batch_lines_s 順序を使用して生成された主キー。

- line_status

  'U'（未処理）に設定する必要がある。

- api_module_id

  hr_api_modules への外部キー。

  次の問合せは、従業員作成のための api_module_id を取得します。

```
SELECT API_MODULE_ID
FROM HR_API_MODULES
WHERE UPPER(MODULE_NAME) = 'CREATE_EMPLOYEE'
AND   UPPER(MODULE_PACKAGE) = 'HR_EMPLOYEE_API';
```

## メタマッパーから生成される PL/SQL パッケージ

メタマッパーは、バッチ・ライン表へのデータのロード、または特定の API の表の内容の検討をユーザーが容易に行えるように、各 API 用に別個のパッケージも生成します。

たとえば、create_employee パッケージ、hrdpp_create_employee には次の 2 つのプロシージャが含まれます。

- insert_batch_lines
- call

### バッチ・ライン挿入プロシージャ

このプロシージャは、バッチ・ライン表へのデータのロードを簡略化するために使用します。

このプロシージャをコールすると、すべてのパラメータ完備で、バッチ・ライン表の中に行が 1 つ作成されます。従業員作成の場合は、次のようなパラメータが含まれます。

```
p_batch_id                     number      in
p_user_sequence                number      in     default
p_link_value                   number      in     default
p_hire_date                    date        in
p_last_name                    varchar2    in
p_sex                          varchar2    in
p_per_comments                 varchar2    in     default
p_date_employee_data_verified  date        in     default
p_date_of_birth                date        in     default
```

```
p_email_address                 varchar2    in     default
p_employee_number               varchar2    in
p_expense_check _send_to_addres varchar2    in     default
p_first_name                    varchar2    in     default
p_known_as                      varchar2    in     default
p_marital_status                varchar2    in     default
p_middle_names                  varchar2    in     default
p_nationality                   varchar2    in     default
p_national_identifier           varchar2    in     default
p_previous_last_name            varchar2    in     default
p_registered_disabled_flag      varchar2    in     default
p_title                         varchar2    in     default
p_attribute1                    varchar2    in     default
p_attribute2                    varchar2    in     default
p_attribute3                    varchar2    in     default
p_attribute4                    varchar2    in     default
p_attribute5                    varchar2    in     default
p_attribute6                    varchar2    in     default
p_attribute7                    varchar2    in     default
p_attribute8                    varchar2    in     default
    ...
    ...
p_resume_exists                 varchar2    in     default
p_resume_last_updated           date        in     default
p_second_passport_exists        varchar2    in     default
p_student_status                varchar2    in     default
p_work_schedule                 varchar2    in     default
p_suffix                        varchar2    in     default
p_person_user_key               varchar2    in
p_assignment_user_key           varchar2    in
p_user_person_type              varchar2    in     default
p_vendor_name                   varchar2    in     default
p_correspondence_language       varchar2    in     default
```

パラメータの数が非常に多いため、この例はすべてのパラメータを示していません。

---

**注意：** このプロシージャには、*p_person_user_key* および *p_assignment_user_key* という 2 つのユーザー・キー値が必要です。これらのキーの値はユーザーが指定する必要があります。Data Pump を使用してオラクル人事管理システム内にレコードを作成する場合、Data Pump は、ユーザーのキー値と、API から生成された人事管理システムのキー値を、ユーザー・キー表に自動的に挿入します。その後の処理では、Data Pump はこれらのキーを使用して、ユーザーの外部システムからのレコードとオラクル人事管理システムのレコードを照合できます。この後の項では、さらに詳しい説明と例を紹介します。

---

### コール・プロシージャ

これは、API をコールして適切なパラメータ値を渡すために、Data Pump プロセスによって実行される実際のラッパー・プロシージャです。このプロシージャでは *p_business_group_id* および *p_batch_line_id* という 2 つの引数が使用されます。

> **注意 :** このプロシージャの直接コールはサポートされていません。このプロシージャを実行するには、Data Pump コンカレント・プロセスを使用する必要があります。

## メタマッパーのヘルプ・プロシージャ

メタマッパー・パッケージには、生成された PL/SQL パッケージおよびビューの名称、そして特定の API のバッチ・ライン表のパラメータ値に関する情報を戻す、ヘルプ・プロシージャ、*hr_pump_meta_mapper help* も含まれています。

ヘルプ・プロシージャには 2 つのパラメータがあります。

- p_module_package

  API PL/SQL パッケージの名称

- p_module_name

  API PL/SQL プロシージャの名称

このプロシージャをコールする前に、サーバー出力をオンに設定する必要があります。

たとえば、*hr_employee_api.create_employee* のヘルプを入手するには、次の SQL*PLUS を使用します。

```
sql> set serveroutput on size 1000000;
sql> execute hr_pump_meta_mapper.help( 'hr_employee_api', 'create_employee' );
```

出力は次のようになります。

```
Generated package: hrdpp_create_employee
Generated view: hrdpv_create_employee

パラメータ名                    タイプ         In/Out  デフォルト？ 参照タイプ
---------------                 -----         ------  --------   -----------
P_HIRE_DATE                     DATE          IN
P_LAST_NAME                     VARCHAR2      IN
P_SEX                           LOOKUP        IN      SEX
P_PER_COMMENTS                  VARCHAR2      IN      DEFAULT
P_DATE_EMPLOYEE
  _DATA_VERIFIED                DATE          IN      DEFAULT
P_DATE_OF_BIRTH                 DATE          IN      DEFAULT
P_EMAIL_ADDRESS                 VARCHAR2      IN      DEFAULT
P_EMPLOYEE_NUMBER               VARCHAR2      IN
```

```
P_EXPENSE_CHECK
  _SEND_TO_ADDRES           LOOKUP       IN      DEFAULT    HOME_OFFICE
P_FIRST_NAME                VARCHAR2     IN      DEFAULT
P_KNOWN_AS                  VARCHAR2     IN      DEFAULT
P_MARITAL_STATUS            LOOKUP       IN      DEFAULT    MAR_STATUS
P_MIDDLE_NAMES              VARCHAR2     IN      DEFAULT
P_NATIONALITY               LOOKUP       IN      DEFAULT    NATIONALITY
P_NATIONAL_IDENTIFIER       VARCHAR2     IN      DEFAULT
P_PREVIOUS_LAST_NAME        VARCHAR2     IN      DEFAULT
P_REGISTERED_DISABLED_FLAG  LOOKUP       IN      DEFAULT    YES_NO
P_TITLE                     LOOKUP       IN      DEFAULT    TITLE
P_WORK_TELEPHONE            VARCHAR2     IN      DEFAULT
P_ATTRIBUTE_CATEGORY        VARCHAR2     IN      DEFAULT
P_ATTRIBUTE1                VARCHAR2     IN      DEFAULT
P_ATTRIBUTE2                VARCHAR2     IN      DEFAULT
P_ATTRIBUTE3                VARCHAR2     IN      DEFAULT
P_ATTRIBUTE4                VARCHAR2     IN      DEFAULT
P_ATTRIBUTE5                VARCHAR2     IN      DEFAULT
P_ATTRIBUTE6                VARCHAR2     IN      DEFAULT
...
P_ASSIGNMENT_SEQUENCE       NUMBER       OUT
P_ASSIGNMENT_NUMBER         VARCHAR2     OUT
P_NAME_COMBINATION_WARNING  BOOLEAN      OUT
P_ASSIGN_PAYROLL_WARNING    BOOLEAN      OUT
P_USER_PERSON_TYPE          VARCHAR2     IN      DEFAULT
P_VENDOR_NAME               VARCHAR2     IN      DEFAULT
P_CORRESPONDENCE_LANGUAGE   VARCHAR2     IN      DEFAULT
...
```

ヘルプ出力の説明は次のとおりです。

- この例では、insert_batch_lines procedure は hrdpp_create_employee.insert_batch_lines です。
- パラメータ名の列は、insert_batch_lines プロシージャ、および生成されたビューに表示されたパラメータの名称を示します。
- パラメータには、ユーザー・キーであることを示す、USER_KEY というタイプが使用されることがあります（詳細は 13-29 ページの「ユーザー・キー値」の項を参照）。たとえば、P_SUPERVISOR_USER_KEY USER_KEY IN DEFAULT です。ユーザー・キー・パラメータのタイプは暗黙で VARCHAR2 になります。
- DATE のパラメータ値は、VARCHAR2 文字列として YYYY/MM/DD の書式で insert_batch_lines プロシージャに渡されます。

**注意：** Data Pump で使用される日付書式は YYYY/MM/DD です。オラクル人事管理システムの日付では、内部日付書式が YYYY/MM/DD HH24:MM:SS です。

- BOOLEAN のパラメータ値は、VARCHAR2 文字列として TRUE または FALSE の値で insert_batch_lines プロシージャに渡されます。
- In/Out の列では、API に渡されたときに PL/SQL IN または IN/OUT であったパラメータ、またはユーザー・キー・パラメータは IN の値で表示されます。パラメータが API PL/SQL OUT パラメータの場合には、In/Out 列の値は OUT です。
- insert_batch_lines プロシージャの引数となるのは IN のパラメータのみです。OUT パラメータは生成されたビューに表示されます。
- デフォルト ? の列では、バッチ・ライン表でパラメータの値が必要とされないものは DEFAULT の値で表示されます。必須パラメータの場合、この列は空白になります。
- 必須パラメータ値は insert_batch_lines プロシージャに渡す必要があります。
- パラメータが参照パラメータの場合、参照タイプの列にはパラメータの参照タイプの名称が入ります。

## バッチ表へのデータのロード

メタマッパーは、各 API 用に特定の PL/SQL パッケージおよびビューを生成します。特に説明がないかぎり、バッチ表にデータをロードするには、これらの PL/SQL インタフェース・プロシージャおよびビューを使用してください。

挿入はインタフェースを通じてのみ実行することが特に重要です。これには 2 つの理由があります。

- PL/SQL プロシージャを使用することにより、ユーザーは基礎スキーマの複雑さから解放されます。
- PL/SQL プロシージャを使用すれば、将来のリリースでスキーマが変更された場合にも影響を受けずにすみます。Data Pump を継続的に使用するつもりであれば、このことが重要になります。

**提案：** ロードするバッチの一部で、従来データの取出しコードをテストおよび検証してください。たとえば、100000 人の詳細をロードする予定であれば、100 人分のサブセットを検証およびロードしてルーチンをテストします。これにより、データ全体をロードする前に、取出しコードに明白な問題があれば、それを識別して解決しておくことができます。

## バッチ・インタフェース表

このインタフェースの設計上の主な目的は、すべてのものを可能なかぎり単純にするということでした。この結果、すべての API のデータ・ロードに関して、Data Pump には 1 つのバッチ・ヘッダーと 1 つのバッチ・ライン表しかありません。メタマッパーにより、各 API 用に特定の列名を含むビューが生成されます。

バッチ・ライン表の各行には、API およびデータ値への参照が含まれます。Data Pump は、パラメータとして渡されたデータを使用して各 API を実行します。

## 処理順序の制御方法

データベースにバッチ・ラインをロードする順序をユーザーが制御する必要のある状況は数多く考えられます。たとえば、個人を作成する前に個人の住所を作成しようとすると、Data Pump のエラーが発生します。

操作を実行する順序を制御するためには、*p_user_sequence* パラメータを使用して手動で順序を設定します。バッチ・ライン表にデータを挿入するときには、このパラメータに対して適切な数値を選択してください。Data Pump はこれらの数値を使用して処理順序を決定します。

### バッチ・ロードの各種方式

バッチ・ラインの処理順序を設定するには、様々な方法があります。

1 つの方法として、別々のバッチで個別のデータをロードすることができます。たとえば、1 つのバッチで個人情報をロードし、2 番目のバッチで住所情報をロードします。

別の方法として、関連する API コールを含むラインを含むバッチを作成できます。たとえば、1 つのバッチの一部として、1 従業員の個人、住所およびアサイメント情報をロードできます。この方法では、パラレル処理オプションを使用する場合、すべてのラインが同じチャンクで処理されるように、*p_link_value* パラメータを使用します。リンクされたグループ内で各種の API コールが正しい順番で行われるように、デフォルト、または *p_user_sequence* パラメータを使用します。

### パラレル実行の場合の処理順序

Data Pump プロセスは、パラレル処理オプションを利用できるように最適化されています。マルチスレッド・プロセスを実行する場合には、バッチ・ラインの順序に関して特別の考慮が必要になります。

Data Pump プロセスをパラレルに実行すると、コンカレント・マネージャは複数のスレッドを生成します。これらのスレッドはそれぞれ、定義された数のバッチ・ラインを処理してから、内容をデータベースにコミットします。ラインの数は CHUNK_SIZE 給与計算処理パラメータで制御されます。詳細は、13-18 ページの「その他のパラメータ」を参照してください。

パラレル処理およびラインのチャンクにより、理論上は、2 つ以上のラインを含むトランザクションは複数のプロセスに分割される可能性があります。これは、*p_user_sequence* パラメータで設定された順序どおりにラインが処理されない可能性があることを意味します。

*p_link_value* パラメータを使用することにより、この状況を避けることができます。このパラメータは、一組のバッチ・ラインを同一のチャンク内で処理する必要があることを Data Pump に指示します。同じスレッドで処理する必要のあるすべてのラインに同じリンク値を使用します。これにより、必要に応じて、1 つのスレッドによって処理される行の数が自動的に増加されます。

---

**注意：** Data Pump をパラレルに実行した場合、パフォーマンスが期待したように向上しない可能性があります。ビジネス・プロセス API をパラレルに実行すると、多量の検証作業により、ロック競合が発生する可能性があることを覚えておいてください。たとえば、個人支払方法 API およびエレメント・エントリ API ではこのような問題が発生することが知られています。

---

## API パラメータのデフォルト値

オラクル人事管理システムの API の設計の一部として、多くのパラメータにはデフォルト値が設定されています。つまり、すべてのパラメータに値を渡す必要なしに、直接に API をコールすることができます。

Data Pump を使用する場合にも、すべてのパラメータに値を指定する必要がないという意味で、同じような機能があります。

次のルールが適用されます。

- バッチ・ライン挿入パラメータに NULL が渡されたか、値が渡されない場合、そしてデフォルトを使用できる場合には、API モジュール自体に適切なデフォルト値が渡されます。
- パラメータに明示的に NULL 値を設定する場合は、<NULL> という特殊な文字列を使用します。NULL 値に更新する場合には、この方法を使用します。

パラメータとして渡されたその他の値はすべて、バッチ・ラインに挿入される値となり、その後適切な API プロセスに渡されます。

# Data Pump プロセスの実行

「レポートと処理の発行」フォームを使用して、Data Pump エンジン・プロセスを起動します。2 つのパラメータが使用されます。

- BATCH NAME

  batch_name は、create_batch_header プロシージャによって挿入されるバッチの 1 つです。

- VALIDATE FLAG

  この値のデフォルト値は No です。これにより、すべての有効なラインがデータベースにコミットされます。

  検証フラグが Yes に設定されている場合には、プロセスは検証モードで実行されます。API はコールされますが、その結果はロールバックされます。データベースに変更をコミットする前にデータを確認して訂正する場合は、このモードを使用します。

> **注意：** Data Pump プロセスを実行する前に、パラレル・スレッドを使用するかどうか、そしてなんらかのロギング・オプションをオンにするかどうかを決定する必要があります。

## パラレル実行

パラレル処理を有効にするには、PAY_ACTION_PARAMETERS で THREADS パラメータの値を設定する必要があります。

スレッド値には開始プロセスが含まれます。つまり、2 という値を設定すると、メイン・エンジン・コードは 1 つのスレーブ・プロセスを開始させるため、コンカレント・プロセスの合計数は 2 になります。パラレルで実行するときには、マスター・プロセスがスレーブ・プロセスの前に終了することがあります。これは正常なことです。

> **注意：** THREADS パラメータは、その他の Oracle Payroll プロセスのパラレル実行も制御します。Data Pump 処理が終了したら、通常の給与計算処理に Data Pump のパラメータが影響しないように、THREADS パラメータをリセットしておく必要があります。

## その他のパラメータ

Data Pump ではこの他に 3 つの給与計算処理パラメータを設定できます。

### CHUNK_SIZE

デフォルト = 10

パラレルで実行するときに、1 つのスレッドで同時にいくつのバッチ API コールを処理するかを制御します。また、1 回のコミットごとの API コールの数も制御します。状況によっては、実際の数字は異なることがあります。たとえば、p_link_value パラメータが設定されているときには、これよりも高くなることがあります。

### MAX_ERRORS_ALLOWED

デフォルト = 20

API コールにおいて、Data Pump エンジン全体が停止するまでに、いくつのエラーが許容されるかを制御します。これは、パラレル・スレッド単位の数値です。

### PUMP_DEBUG_LEVEL

Data Pump プロセスから生成されたエラーを追跡するためにロギングをオンにするには、このパラメータを使用します。このパラメータの有効な値の一覧は、13-19 ページの「ロギング・オプション」を参照してください。

## 実行状況の確認

Data Pump はコンカレント・プロセスとして実行されるため、「コンカレント要求の表示」ウィンドウを使用して、いつでもその状況を確認できます。プロセス全体が失敗した場合にのみ、コンカレント・マネージャから失敗が報告されます。通常この状況は、エラー数が MAX_ERRORS_ALLOWED パラメータで設定された値を超えたために発生します。

---

**注意：** コンカレント・プロセスが正常終了した場合にでも、プロセスでデータ・エラーが発生している可能性があります。バッチ・ライン・エラーは必ず調べてください。

---

# エラーの検索と修正

この項では、Data Pump プロセスから生成されたエラーを追跡するために使用するロギング・オプションと、これらの処理方法に関するヒントについて説明します。

## ロギング・オプション

Data Pump のロギング・オプションを有効にするには、PAY_ACTION_PARAMETERS 表の PUMP_DEBUG_LEVEL パラメータに適切な値を挿入します。

**注意：** ロギングをオンにすると、Data Pump プロセス全体のパフォーマンスに必ず影響があります。ロギングは、問題が発生した場合に問題の追跡を行うためにのみ使用してください。問題が解決した後でロギングをオフにすることも忘れないでください。

PUMP_DEBUG_LEVEL の有効な値は次のとおりです。

**提案：** 最初の 3 つのオプションが最も有用であるはずです。

| オプション | 説明 |
|---|---|
| AMD | API モジュール・デバッグ（API からトレース出力を可能にする） |
| RRP | 行範囲処理ロギング（各作業単位または範囲で発生したエラーの数を記録する） |
| GID | get_id 関数失敗情報（ユーザー値を ID にマップする関数の失敗を記録する） |
| MSG | 特定のロギング・メッセージを出力する |
| ROU | ルーティング情報（プロシージャのエントリおよび終了） |
| WCD | ラッパー・キャッシュ・デバッグ・ロギング |
| STK | スタック・ダンプ・ロギング（失敗時に情報をトレース） |
| EXT | 終了情報（成功時に情報をトレース） |
| RRI | 行範囲挿入ロギング |
| BLI | バッチ・ライン情報（処理中のバッチ・ラインのバッチ・ライン番号を出力） |
| CLF | コンカレント・ログ・ファイル（コンカレント・マネージャ・ログ・ファイルに送られる、MSG オプション付きのロギング・メッセージ出力） |

コロンで区切って値を連結すれば、これらのオプションを必要なだけ組み合せることができます。例:

```
update pay_action_parameters
set    parameter_value = 'MSG:RRI:RRP'
where  parameter_name  = 'PUMP_DEBUG_LEVEL';
```

## ロギング出力の表示方法

ロギング・オプションを有効にすると、実行中のすべてのスレッドに関して出力が作成されます。この出力を表示するには PYUPIP コマンドを使用します。

このコマンドを使用するには、ロギング対象のコンカレント・プロセスの ID を知る必要があります。オンラインからは、「要求の表示」ウィンドウを使用してコンカレント要求 ID を調べることができます。また、HR_PUMP_REQUESTS 表から問い合せることもできます。実行中のプロセスごとに 1 行が挿入されます。例:

```
select * from hr_pump_requests;
```

典型的な出力は次のようになります。

```
BATCH_ID    REQUEST_ID      PROCESS_TYPE
----------- --------------- -----------------
8437        98533           MASTER
8437        98534           SLAVE
```

ここでは、2 つのプロセスが実行中であり、request_id 値は 98533 および 98534 であることを知ることができます。

別個のコマンド行ウィンドウで出力をトレースするには PYUPIP を使用します。例:

```
PYUPIP <user/password>@database REQID98533
PYUPIP <user/password>@database REQID98534
```

**注意:** 複数のスレッドを実行している場合は、すべてのスレッドをトレースする必要があります。そうしないと、データベース・トレース・パイプが満杯になったときに処理が停止します。トレース時にはスレッドを 1 つのみ実行することをお薦めします。

## バッチ・ラインのエラーの検索方法

処理中にエラーが発生すると、Data Pump は HR_PUMP_BATCH_EXCEPTIONS 表の中に行を生成します。このリリースでは、この情報を表示するために SQL*PLUS を使用する必要があります。

また、SQL*PLUS を使用して、HR_PUMP_BATCH_LINES の中で、LINE_STATUS の値が E - エラーである行を問い合せることができます。

---

**注意：** 検証モードでは、LINE_STATUS は API コールが成功すると、V- 検証済に設定されます。更新モードでは、LINE_STATUS は API コールが成功すると、C- 接続に設定されます。

---

## エラーの原因調査

調査方法は、エラーのタイプと、その発生場所によって異なります。エラーによっては、原因を究明するために API やオラクル人事管理システム・アプリケーションの使用経験が必要になることがあります。

Data Pump の場合のアドバイスをいくつか紹介します。

- 最初に HR_PUMP_BATCH_EXCEPTIONS 表の列でエラーの発生したバッチ・ラインを確認します。これを使用して、バッチ・ライン自体のパラメータおよび値を調べます。
- 一般的なエラーの 1 つに「データが見つかりません」というものがあります。このエラーは多くの場合、ユーザー内容を ID 値に変換するためにコールされた関数の 1 つにエラーがあるために発生します。この場合は、例外表を調べてもエラーの正確な原因を知ることはできません。GID ロギング値を使用するとさらに詳しい情報を入手できます。障害が発生した場合には、失敗した関数の名称と、渡された引数値がトレースに表示されます。
- AMD ロギング値は、問題を追跡する助けとなります。これにより API モジュール自体のロギングが起動され、調査する必要のある豊富な出力が提供されます。
- データ・ロードの順序の誤りも、エラーの一般的な原因の 1 つです。たとえば、個人をロードする前に個人の住所をロードした場合です。パラレル処理を使用した場合に、複数のバッチ・ラインを関連付けるための LINK_VALUE を使用していない場合には、関連エラーも発生する可能性があります。
- 検証モードで実行した場合には、他のチャンクの結果に依存しないようにバッチをチャンクに分割していないと、順序エラーが発生します。このエラーは、単一のスレッドで検証が行われた場合にも発生します。これは、1 つのチャンクの API の結果は、ロールバック・セグメントを解放するためにロールバックされるためです。これは、ロードの実行を制御するために *p_link_value* パラメータを使用するもう 1 つの理由になります。

## エラーの修正方法

エラーの最も一般的な原因は、insert_batch_lines プロシージャによって正しくない値がロードされたことです。これを修正する必要があります。

## ビューを使用したデータの訂正

HR_PUMP_BATCH_LINES の HRDPV_views を使用して適切な列の中の値を訂正します。これらのビューでは通常の update 文を使用できるため、データの問題をより簡単に修正できます。

**警告：** 問題データを変更するためにビューを使用するときには、HR_PUMP_BATCH_LINES 表の LINE_STATUS は変更しないでください。Data Pump エンジンは処理にこれを使用します。

**注意：** HR_PUMP_BATCH_LINES のビューは、その API に対して生成された行のみを表示します。API_MODULE_ID 列を正しくない値で更新しようとすると、ORA-1402 エラーにより操作は失敗します。ユーザーがビューを使用して、そのビューでは選択できない行を生成しないように、ビューは WHERE 句に WITH CHECK OPTION を指定して生成されています。

（生成されたビューを使用して HR_PUMP_BATCH_LINES に行を挿入する場合にも、同じ警告が当てはまります。）

## Data Pump プロセスの再実行

問題を訂正したら、同じバッチ名を使用して Data Pump プロセスを再度発行して、バッチを再実行できます。すべてのラインが正常終了するまで、何度でもプロセスを発行できます。ステータスが E - エラー、U- 未処理、または V - 検証済のバッチ・ラインは自動的に再処理されます。

例外表から表を削除するための処理は必要ありません。Data Pump は自動的にこれを処理します。

前回の Data Pump 実行で検証されたラインは、Data Pump を検証モードで実行した場合にも再処理されます。これは、関連する API コールの結果が、前回の実行でロールバックされている可能性があるからです。再処理されないのは、ステータスが C - 完了であるラインのみです。

# データのパージ

現在の Data Pump では、例外表の中の行の自動削除を除いては、バッチ表からデータを自動的に削除するパージ・プロセスは提供されていません。それ以外のすべての場合に、ユーザーはどのデータをいつパージする必要があるかを考えておく必要があります。

**注意：** ユーザー・キー値表からデータをパージするときには、特に注意が必要です。たとえば、アサイメントおよび個人ユーザー・キーを削除すると、最初に add_user_key プロシージャを使用して、パージされたユーザー・キーを再作成しないかぎり、その従業員のサブ・アサイメントは作成できなくなります。このため、USER_KEYS 表は Data Pump 処理が完了した場合にのみパージするようにお薦めします。

## パージ方法

常に次の処理から開始する必要があります。

```
TRUNCATE TABLE HR_PUMP_REQUESTS;
TRUNCATE TABLE HR_PUMP_RANGES;
```

### 全行の単純なパージ

ステータスに関係なく、すべての行をパージするには、次のようにします。

```
TRUNCATE TABLE HR_PUMP_BATCH_EXCEPTIONS;
TRUNCATE TABLE HR_PUMP_BATCH_LINE_USER_KEYS;
TRUNCATE TABLE HR_PUMP_BATCH_LINES;
TRUNCATE TABLE HR_PUMP_BATCH_HEADERS;
```

### すべての終了行のパージ

これはより複雑です。すべてのロードが終了しないかぎりデータをパージしないでください。これにより、まだ必要な行がパージされる危険性を防止できます。次の処理を実行します。

- HR_PUMP_BATCH_LINES.LINE_STATUS 列を使用して、どの行が終了している、そのためパージ可能であるかを指示します。
  - C というステータスを探します。もちろん、バッチ内のすべての行のステータスが C である場合は、バッチ中のすべての行を単純にパージします。
- 次に示す順序で、下記の表の中の適切な行をすべて削除します。
  - HR_PUMP_BATCH_EXCEPTIONS
  - HR_PUMP_BATCH_LINE_USER_KEYS
  - HR_PUMP_BATCH_LINES

HR_PUMP_BATCH_LINES の中のすべての行が削除されたら、HR_PUMP_BATCH_HEADER 表から適切なバッチを削除します。

## サンプル・コード

この項では、バッチ・ライン・プロシージャのコール方法を示すサンプル・コードを紹介します。

この例は API コール用のデータを生成する架空のものです。しかし、ここでは、次のような処理を行う多数のバッチ・ラインを作成するために、Data Pump をどのように準備するかが示されています。

- 従業員を作成する
- 従業員の住所を作成する
- デフォルト・アサイメント基準を更新する
- サブ・アサイメントを作成する

この例は、各従業員およびアサイメントの個別のトランザクションが同じスレッドで処理されるようにするために *p_link_value* を使用する方法も示しています。

```
----------------------- start of example -----------------------
create or replace package hrdp_cre_emp as
procedure hrdp_cre_emp (p_start in number, p_end in number);
end hrdp_cre_emp;
/
create or replace package body hrdp_cre_emp as
/*
 *  Insert a number of batch lines in preparation for
 *  running the data pump engine, which will then
 *  - create an employee
 *  - create an address for the employee
 *  - update the criteria of the default assignment
 *  - create a secondary assignment
 */
procedure hrdp_cre_emp (p_start in number, p_end in number) is
   l_last_name    varchar2(40);
   l_hire_date    date;
   l_birthday     date;
   l_first_name   varchar2(40);
   l_asgno        varchar2(40);
   -- These are the 'out' values.
   l_special_ceiling_step_id number;
   l_person_user_key             varchar2(100);
   l_address_user_key            varchar2(100);
   l_assignment_user_key         varchar2(100);
   l_assignment_user_key2        varchar2(100);
```

```
   l_link_value                  number;
   l_commit_count number;
   l_commit_limit number;
   l_emp_count    number;
   l_address_line1 varchar2(256);
begin
   l_commit_limit := 10;   -- commit after every 10 employees.
   l_commit_count := 0;
   l_first_name   := 'David';
   l_hire_date    := to_date('1997/12/01', 'YYYY/MM/DD');
   l_birthday   := to_date('1970/01/01', 'YYYY/MM/DD');
   l_link_value := 0;
   for emp_count in p_start..p_end loop
      -- Prepare to create an employee.
     l_last_name := 'DUMP' || lpad(emp_count, 5, '0');
     l_person_user_key     := l_last_name || ' : PER USER KEY';
     l_assignment_user_key := l_last_name || ' : ASG USER KEY';
     l_address_user_key := l_last_name || ' : ADDR USER KEY';
     l_address_line1 := to_char(emp_count) || ', Union Square';
     hr_utility.trace('Last Name : ' || l_last_name);
     -- Allow linking together so that these API calls process
     -- by the same thread.
     l_link_value := l_link_value + 1;
     hrdpp_create_employee.insert_batch_lines
     (
         p_batch_id                  => 3,
         p_user_sequence             => null,
         p_link_value                => l_link_value,
         p_person_user_key           => l_person_user_key,
         p_assignment_user_key       => l_assignment_user_key,
         p_hire_date                 => l_hire_date,
         p_last_name                 => l_last_name,
         p_sex                       => 'Male',
         p_employee_number           => null,
         p_per_comments              => 'Comments for : ' || l_last_name,
         p_date_of_birth             => l_birthday,
         p_email_address             => 'somebody@us.oracle.com',
         p_first_name                => l_first_name,
         p_user_person_type          => 'Employee'
     );
     -- Create an address for the person.
     hrdpp_create_us_person_address.insert_batch_lines
     (
        p_batch_id                  => 3,
        p_user_sequence             => null,
        p_link_value                => l_link_value,
        p_effective_date            => l_hire_date,
```

```
   p_primary_flag              => 'Yes',
   p_date_from                 => l_hire_date,
   p_address_type              => 'Home',
   p_address_line1             => l_address_line1,
   p_city                      => 'Golden Valley',
   p_county                    => 'Los Angeles',
   p_state                     => 'California',
   p_zip_code                  => '91350',
   p_country                   => 'US',
   p_person_user_key           => l_person_user_key,
   p_address_user_key          => l_address_user_key
);
-- Let's update some criteria.
l_special_ceiling_step_id := hr_api.g_number;
hrdpp_update_emp_asg_criteria.insert_batch_lines
(
   p_batch_id                  => 3,
   p_user_sequence             => null,
   p_link_value                => l_link_value,
   p_effective_date            => l_hire_date,
   p_datetrack_update_mode     => 'CORRECTION',
   p_assignment_user_key       => l_assignment_user_key,
   p_payroll_name              => 'Monthly',
   p_special_ceiling_step_id   => l_special_ceiling_step_id
);
l_assignment_user_key2 := l_assignment_user_key || '2';
hrdpp_create_secondary_emp_asg.insert_batch_lines
(
   p_batch_id                  => 3,
   p_user_sequence             => null,
   p_link_value                => l_link_value,
   p_assignment_user_key       => l_assignment_user_key2,
   p_person_user_key           => l_person_user_key,
   p_effective_date            => l_hire_date,
   p_assignment_number         => l_asgno,
   p_comments                  => 'asg created by data pump',
   p_organization_name         => 'Setup Business Group',
   p_grade_name                => 'faz1',
   p_job_name                  => 'TEST',
   p_payroll_name              => 'Monthly'
);
l_hire_date    := l_hire_date + 1;
l_commit_count := l_commit_count + 1;
if(l_commit_count = l_commit_limit) then
   -- Commit after so many employees.
   hr_utility.trace('Commit after ' || l_commit_limit || ' employees.');
   commit;
```

```
        l_commit_limit := 1;
      end if;
   end loop;
end hrdp_cre_emp;
/
```

# 生成済インタフェースの使用方法に関する注意

メタマッパー・プロセスは、各 API 用に PL/SQL パッケージおよびビューを生成します。この項では、これらを使用する場合の注意事項を説明します。

## 名称または値からシステム ID を検索する

API を使用するときには、多くのパラメータで参照コードおよび代理主キーを指定する必要があります。例:

```
...
p_sex        => 'M',
p_payroll_id => 13456,
...
```

Data Pump がない場合には、各 API について、外部システムからオラクル人事管理システムのシステム ID に値を変換する追加コードを作成することが必要になります。

しかし Data Pump の場合には、サポート済 API のそれぞれについて、ユーザー名または値を参照およびシステム ID に自動的に変換するシステム定義のプロシージャが一組提供されています。例:

```
...
p_sex                  => 'Male',
p_payroll_name => 'Monthly Payroll',
...
```

---

**注意:** 参照パラメータの場合には、参照内容または参照コード自体を使用できます。参照タイプ以外の ID の場合には、代替パラメータを使用できます。

---

**例外** パラメータ値の名称の使用方法については、3 つの大きな例外があります。

- フレックスフィールド属性パラメータ
- PL/SQL IN/OUT パラメータ
- 国別仕様の参照パラメータ

### フレックスフィールド属性パラメータ

ほとんどの API プロセスには、P_SEGMENT18 または P_ATTRIBUTE20 などの名称を持つフレックスフィールド属性パラメータがあります。Data Pump はユーザーの特定のシステムでこれらの値がどのようにマッピングされるか判断できないため、値の変換はサポートされていません。

つまり、ユーザーには、正しい参照コードまたはその他の値を適切に渡す責任があります。

### PL/SQL IN/OUT パラメータ

API が複数の異なる処理の組合せを実行する場合、ユーザーは、ユーザー内容ではなく適切な ID またはコード値をパラメータに指定する必要があります。このような項目の値は Data Pump の実行前に導出できるため、大きな問題にはならないはずです。

たとえば、hr_assignment_api.update_emp_asg では、他の API ではユーザー・キーにする必要があるとしても、p_special_ceiling_step_id は ID として渡す必要があります。

**注意：** API の PL/SQL IN/OUT パラメータのユーザー・キーを提供することはできません。なぜなら、特定の API をコールする Data Pump コードは、API コールの前にユーザー・キーが存在していたかどうかを知ることはできず、ユーザー・キーを作成するのか、ユーザー・キーの ID 値が API コール後に更新されるのかを判断できないからです。

多くの API が出力パラメータとして comment_id を生成します。しかし、comment_id のユーザー・キー値を指定する必要はありません。これにより、意味のない多数のユーザー・キーが生成されることを防止できます。

**注意：** comment_id ユーザー・キーは、エレメント・エントリ作成および更新 API の comment_id には必須です。エレメント・エントリ API コールに必要な場合は、これらのユーザー・キーを追加する必要があります。

### 国別仕様の参照パラメータ

国別仕様の API コールでの特定の参照が、汎用ビジネス・プロセス API コールの汎用パラメータに対応するときには、国別仕様ビジネス・プロセス API コールでも同様の状況が発生します。

たとえば、「hr_person_address_api.create_person_address」API の中の *p_region_1* パラメータは、「hr_person_address_api.create_gb_person_address」API の中の *p_county* 参照パラメータに対応します。

Data Pump を通じて UK 住所の hr_person_address_api.create_person_address をコールするときには、*p_region_1* パラメータとして 'GB_COUNTY' 参照コードを渡す必要があります。また、hr_person_address_api.create_gb_person_address を使用した場合は 'GB_COUNTY' 参照内容を使用できます。

**注意：** 可能な場合は国別仕様の API を使用してください。

## ユーザー・キー値

外部システムからオラクル人事管理システムにデータをマッピングするときには、Oracle エンティティの ID 値を、論理一意キーまたは名称から導出できない場合があります。この例としては個人、アサイメント、住所があります。個人の一意識別子を考えてみます。個人を識別するのは、不可能とは言わないまでも非常に困難です。理論上は、同じ姓、名、性別、生年月日、婚姻区分などを共有する複数の人物が存在することは可能です。

エンティティに論理キーがなく、その代理 ID をそのコンポーネント・エンティティのいずれの名称からも簡単に導出できない場合には、同じような問題が発生します。たとえば、給与、役職、職階などのコンポーネントの名称を単純に調べただけで、一意のエレメント・リンクを識別することは容易ではありません。

または、Oracle Applications 製品に固有の抽象エンティティなど、ID 値を使用してしか識別できないエンティティもあります。たとえば、ID_FLEX_NUM です。

Data Pump が提供する解決方法とは、ユーザーが「ユーザー・キー」値を設定できるようにすることです。この値は一意の文字列である必要があります。外部システムからの一意の ID を使用することもできますし、複数の値を連結することもできます。たとえば、個人の氏名と、従来システムからの既存の従業員番号を連結して、個人のユーザー・キーとすることができます。次のようになります。

```
p_person_user_key => 'Joe Bloggs' || '2345',  -- name + emp no
```

すべてのパラメータで、ユーザー・キー値は 'user_key' で終了する名称で定義する必要があります。Data Pump はこれらのユーザー・キー値を使用して、オラクル人事管理システム内のレコードの ID を識別します。

**注意**ユーザー・キー値はすべてのエンティティを通じて一意のものにする必要があります。たとえば、個人ユーザー・キー値に 'SMITH1001' を使用し、アサイメント・ユーザー・キー値にも 'SMITH1001' を使用することはできません。

ほとんどの場合は、各システム ID に対して 1 つのユーザー・キー値を使用します。しかし、Data Pump の場合は、同じシステム ID に対して複数の異なるユーザー・キーを定義できます。複数の異なる外部システムからデータをロードする場合に、一意キーが照合されない場合は、これが重要になります。

ユーザー・キーは HR_PUMP_BATCH_LINE_USER_KEYS 表の中に行として保存されます。

### ユーザー・キー値の作成

ユーザー・キーは 2 つの方法のいずれかで作成されます。

- Data Pump で新しいユーザー・キーを挿入

  Data Pump を使用する場合には、いくつかの API パラメータのユーザー・キーを指定する必要があります。新規レコードを作成する API のコールが成功すると、Data Pump は、ユーザーが指定した名称と、API から戻されたシステム ID 値を使用して、ユーザー・キー表に新しい行を挿入します。戻された ID 値は API の PL/SQL OUT パラメータです。

- 手動で新しいユーザー・キーを挿入

  外部システムからすでにデータをロードしてある場合、または同じシステム ID に対して複数のユーザー・キーを作成する場合には、*add_user_key* ユーティリティ・プロシージャを使用して、手動で HR_PUMP_BATCH_LINE_USER_KEYS に行を挿入できます。

ユーザー・キーを作成したら、他の API でも同じキーを使用して、既存エンティティの更新や別のエンティティの指定を行うことができます。たとえば、2 つの個人ユーザー・キーを使用して続柄を指定できます。

# Data Pump で使用可能なユーティリティ・プロシージャ

この項では、Data Pump から提供されるユーティリティ・プロシージャを紹介します。

すべてのプロシージャは HR_PUMP_UTILS パッケージに入っています。

## create_batch_header

パラメータ :

```
    p_batch_name            : 一意のバッチ名
    p_business_group_name   : ビジネス・グループの名称（オプション）
    p_reference             : ユーザー参照値（オプション）
Returns
    The hr_pump_batch_headers.batch_id.
```

説明 :

バッチ・ヘッダー行を作成します。直接挿入ではなく、行の作成に使用してください。

このプロシージャのコール例は次のようになります。

```
  declare
     l_batch_id number;
  begin
     l_batch_id := hr_pump_utils.create_batch_header
                 ('Employees for Dept 071', 'AKA Enterprises');
  end;
```

## add_user_key

プロシージャ :add_user_key

パラメータ :

```
    p_user_key_value        : 一意のユーザー・キー名
    p_unique_key_id         : ユーザー・キーに関連する ID
```

説明 :

Data Pump API コールで使用するユーザー・キーを作成します。add_user_key は、ID 値で参照するオブジェクトが Data Pump により作成されていない場合に、ユーザー・キーを追加するのに使用します。これは、Data Pump によりコールされる API がユーザー・キー・パラメータを必要とするが、オブジェクトにユーザー・キーを作成する API が存在しない場合、または Data Pump が使用可能となる前にオブジェクトが作成されている場合に発生する可能性があります。

### modify_user_key

プロシージャ :modify_user_key
パラメータ :
p_user_key_value : ユーザー・キーの変更を識別する一意のユーザー・キー値
p_new_user_key_value : 一意の新規ユーザー・キー値
p_unique_key_id : ユーザー・キーに関連する新規 ID
説明 :
modify_user_key の主な目的は、add_user_key により作成された不正なユーザー・キーを修正することです。p_new_user_key_value または p_unique_key_id のどちらかが NULL の場合、対応する列のユーザー・キーは更新されません。

## 表およびビューの説明

この項では、Data Pump で使用される特定の表およびビューについてさらに詳しく説明します。

### HR_API_MODULES

Data Pump でサポートされる API モジュール

| 名称 | 説明 |
|---|---|
| API_MODULE_ID | 順序で生成された一意の ID。 |
| API_MODULE_TYPE | 次で表される API のタイプ。<br>'RH' - 行ハンドラ<br>（Data Pump では重要でありません）。<br>'BP' - ビジネス・プロセス API。<br>'AI' - 代替インタフェース API。 |
| MODULE_NAME | API プロシージャ名。 |
| MODULE_PACKAGE | モジュール・タイプが 'BP' または 'AI' の場合の API パッケージ名。 |

### HR_PUMP_BATCH_LINE_USER_KEYS

この表は、外部システムとオラクル人事管理システムの間のキー・マッピングを含みます。これらのキーは、バッチ・ライン表の中の 1 つのフィールドからオラクル人事管理システムの中のレコードを一意に識別することが困難な場合に、特定のエンティティで必要となるものです。たとえば、外部システムから氏名および国別識別子を使用して、オラクル人事管理システムの Person ID にマップすることができます。

この表は、ユーザーがオラクル人事管理システムに新しいレコードを作成すると、Data Pump プロセスによって自動的に作成されます。たとえば、従来データをロードした場合です。すでに従来データをロードしてある場合には、ユーザーはこの表に新しい行を挿入できます。

オラクル人事管理システムの同じ unique_key_id に複数の外部キーをマッピングできます。たとえば、異なる一意 ID を持つ、外部の給与システムおよび外部の福利厚生システムからのデータを、Oracle Human Resources とインタフェースする場合です。

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| USER_KEY_ID | NOT NULL | NUMBER(9) | |
| BATCH_LINE_ID | | NUMBER(9) | |
| USER_KEY_VALUE | NOT NULL | VARCHAR2(240) | レコードを識別するための<br>ユーザー定義キー。 |
| UNIQUE_KEY_ID | NOT NULL | NUMBER(15) | オラクル人事管理システムの<br>一意のキー。 |
| LAST_UPDATE_DATE | DATE | | |
| LAST_UPDATED_BY | | NUMBER(15) | |
| LAST_UPDATE_LOGIN | | NUMBER(15) | |
| CREATED_BY | | NUMBER(15) | |
| CREATION_DATE | | DATE | |

## HR_PUMP_BATCH_HEADERS

この表は、Data Pump のバッチ・ヘッダー情報を含みます。BATCH_NAME は Data Pump コンカレント・プロセスのパラメータです。

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| BATCH_ID | NOT NULL | NUMBER(9) | |
| BATCH_NAME | NOT NULL | VARCHAR2(80) | バッチの一意な名称。 |
| BATCH_STATUS | NOT NULL | VARCHAR2(30) | ステータスは 'ACTION STATUS'<br>参照タイプを使用してデコード可能。 |
| REFERENCE | | VARCHAR2(80) | |
| BUSINESS_GROUP_NAME | | VARCHAR2(80) | |
| LAST_UPDATE_DATE | | DATE | |
| LAST_UPDATE_LOGIN | | NUMBER(15) | |
| LAST_UPDATED_BY | | NUMBER(15) | |
| CREATED_BY | | NUMBER(15) | |
| CREATION_DATE | | DATE | |

## HR_PUMP_BATCH_LINES

この表は、Data Pump によってロードされる個々のバッチ・ラインを含みます。

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| BATCH_LINE_ID | NOT NULL | NUMBER(9) | 順序で生成された ID。 |
| BATCH_ID | NOT NULL | NUMBER(9) | HR_PUMP_BATCH_HEADERS への外部キー。HR_PUMP_BATCH_HEADERS |
| API_MODULE_ID | NOT NULL | NUMBER(9) | HR_API_MODULES への外部キー。HR_API_MODULES |
| LINE_STATUS | NOT NULL | VARCHAR2(1) | この API のロード・ステータス。'U' 未処理（初期）'V' - 検証済だが、レコードは未コミット。'C' - 完了およびレコードがコミット済。'E' - エラー。 |
| PROCESS_SEQUENCE | | NUMBER(9) | |
| USER_SEQUENCE | | NUMBER(9) | |
| LINK_VALUE | | NUMBER | |
| PVAL001 | | VARCHAR2(2000) | |
| PVAL002 | | VARCHAR2(2000) | |
| PVAL003 | | VARCHAR2(2000) | |
| PVAL004 | | VARCHAR2(2000) | |
| PVAL005 | | VARCHAR2(2000) | |
| PVAL006 | | VARCHAR2(2000) | |
| PVAL007 | | VARCHAR2(2000) | |
| PVAL008 | | VARCHAR2(2000) | |
| PVAL009 | | VARCHAR2(2000) | |
| PVAL010 | | VARCHAR2(2000) | |
| PVAL230 | | VARCHAR2(2000) | |
| PLONGVAL | | LONG | |

## HR_PUMP_BATCH_EXCEPTIONS

例外情報を含みます。

| 名称 | 説明 |
|---|---|
| EXCEPTION_SEQUENCE | 順序で生成された一意の ID。 |
| EXCEPTION_LEVEL | 'MESSAGE_LEVEL' 参照を使用したデコード。 |
| SOURCE_ID | BATCH_ID または BATCH_LINE_ID。 |
| SOURCE_TYPE | SOURCE_ID が次を含むことを示す。'BATCH_HEADER' :BATCH_ID 'BATCH_LINE' :BATCH_LINE_ID |
| EXCEPTION_TEXT | 例外のテキスト。 |

## HRDPV_CREATE_EMPLOYEE

| 名称 | NULL? | タイプ |
|---|---|---|
| BATCH_ID | NOT NULL | NUMBER(9) |
| BATCH_LINE_ID | NOT NULL | NUMBER(9) |
| API_MODULE_ID | NOT NULL | NUMBER(9) |
| LINE_STATUS | NOT NULL | VARCHAR2(1) |
| USER_SEQUENCE | | NUMBER(9) |
| LINK_VALUE | | NUMBER |
| P_HIRE_DATE | | VARCHAR2(2000) |
| P_LAST_NAME | | VARCHAR2(2000) |
| P_SEX | | VARCHAR2(2000) |
| P_PER_COMMENTS | | VARCHAR2(2000) |
| P_DATE_EMPLOYEE_DATA_VERIFIED | | VARCHAR2(2000) |
| P_DATE_OF_BIRTH | | VARCHAR2(2000) |
| P_EMAIL_ADDRESS | | VARCHAR2(2000) |
| P_EMPLOYEE_NUMBER | | VARCHAR2(2000) |
| P_EXPENSE_CHECK_SEND_TO_ADDRES | | VARCHAR2(2000) |
| P_FIRST_NAME | | VARCHAR2(2000) |
| P_KNOWN_AS | | VARCHAR2(2000) |
| P_MARITAL_STATUS | | VARCHAR2(2000) |
| P_MIDDLE_NAMES | | VARCHAR2(2000) |
| P_NATIONALITY | | VARCHAR2(2000) |
| P_NATIONAL_IDENTIFIER | | VARCHAR2(2000) |
| P_PREVIOUS_LAST_NAME | | VARCHAR2(2000) |
| P_REGISTERED_DISABLED_FLAG | | VARCHAR2(2000) |
| P_TITLE | | VARCHAR2(2000) |
| P_WORK_TELEPHONE | | VARCHAR2(2000) |
| P_ATTRIBUTE_CATEGORY | | VARCHAR2(2000) |
| P_ATTRIBUTE1 | | VARCHAR2(2000) |
| P_ATTRIBUTE2 | | VARCHAR2(2000) |
| P_ATTRIBUTE3 | | VARCHAR2(2000) |
| ... | | |
| P_ATTRIBUTE30 | | VARCHAR2(2000) |
| P_PER_INFORMATION_CATEGORY | | VARCHAR2(2000) |
| P_PER_INFORMATION1 | | VARCHAR2(2000) |
| P_PER_INFORMATION2 | | VARCHAR2(2000) |
| P_PER_INFORMATION3 | | VARCHAR2(2000) |
| ... | | |
| P_PER_INFORMATION30 | | VARCHAR2(2000) |
| P_BACKGROUND_CHECK_STATUS | | VARCHAR2(2000) |
| P_BACKGROUND_DATE_CHECK | | VARCHAR2(2000) |
| P_BLOOD_TYPE | | VARCHAR2(2000) |
| P_FAST_PATH_EMPLOYEE | | VARCHAR2(2000) |
| P_FTE_CAPACITY | | VARCHAR2(2000) |
| P_HONORS | | VARCHAR2(2000) |
| P_INTERNAL_LOCATION | | VARCHAR2(2000) |

| | |
|---|---|
| P_LAST_MEDICAL_TEST_BY | VARCHAR2(2000) |
| P_LAST_MEDICAL_TEST_DATE | VARCHAR2(2000) |
| P_MAILSTOP | VARCHAR2(2000) |
| P_OFFICE_NUMBER | VARCHAR2(2000) |
| P_ON_MILITARY_SERVICE | VARCHAR2(2000) |
| P_PRE_NAME_ADJUNCT | VARCHAR2(2000) |
| P_PROJECTED_START_DATE | VARCHAR2(2000) |
| P_RESUME_EXISTS | VARCHAR2(2000) |
| P_RESUME_LAST_UPDATED | VARCHAR2(2000) |
| P_SECOND_PASSPORT_EXISTS | VARCHAR2(2000) |
| P_STUDENT_STATUS | VARCHAR2(2000) |
| P_WORK_SCHEDULE | VARCHAR2(2000) |
| P_SUFFIX | VARCHAR2(2000) |
| P_PERSON_USER_KEY | VARCHAR2(2000) |
| P_ASSIGNMENT_USER_KEY | VARCHAR2(2000) |
| P_PER_OBJECT_VERSION_NUMBER | VARCHAR2(2000) |
| P_ASG_OBJECT_VERSION_NUMBER | VARCHAR2(2000) |
| P_PER_EFFECTIVE_START_DATE | VARCHAR2(2000) |
| P_PER_EFFECTIVE_END_DATE | VARCHAR2(2000) |
| P_FULL_NAME | VARCHAR2(2000) |
| P_PER_COMMENT_ID | VARCHAR2(2000) |
| P_ASSIGNMENT_SEQUENCE | VARCHAR2(2000) |
| P_ASSIGNMENT_NUMBER | VARCHAR2(2000) |
| P_NAME_COMBINATION_WARNING | VARCHAR2(2000) |
| P_ASSIGN_PAYROLL_WARNING | VARCHAR2(2000) |
| P_USER_PERSON_TYPE | VARCHAR2(2000) |
| P_VENDOR_NAME | VARCHAR2(2000) |
| P_CORRESPONDENCE_LANGUAGE | VARCHAR2(2000) |

## PAY_ACTION_PARAMETERS

| 名称 | NULL? | タイプ |
|---|---|---|
| PARAMETER_NAME | NOT NULL | VARCHAR2(30) |
| PARAMETER_VALUE | NOT NULL | VARCHAR2(80) |

# 14

# Oracle Generic Third Party Payroll Backfeed

## Oracle Generic Third Party Payroll Backfeed（オラクル以外の給与システムのバックフィード）

この章では、Oracle Generic Third Party Payroll Backfeed の理解と使用のために必要な事項について説明します。この情報を理解するためには、次の事項を含めて、オラクル人事管理システム製品アーキテクチャの機能上および技術上の知識が必要となります。

- オラクル人事管理システムのデータ・モデル
- API 計画、および API を直接コールする方法
- PL/SQL のコーディング方法
- コンカレント・プロセスの実行を制御する HRMS パラメータ
- Data Pump の使用方法および構成方法

### 内容

この章は次の項で構成されています。


- 概要 : 14-2 ページ

  Oracle Generic Third Party Payroll Backfeed の概要について説明します。

- Oracle Generic Third Party Payroll Backfeed の設定 : 14-3 ページ

  オラクル以外の給与システムのバックフィードの高度な設定のステップについて説明します。各ステップについては次の項で詳しく説明します。

  - Oracle Generic Third Party Payroll Backfeed のインストール : 14-4 ページ
  - 支払情報 : 14-6 ページ
  - バランス・タイプ : 14-6 ページ

  - Data Pump の設定 : 14-8 ページ
  - データ・アップローダの設定 : 14-10 ページ
- バックフィードを使用した給与計算結果のアップロード : 14-14 ページ

  オラクル以外の給与システムのバックフィードの高度な使用のステップについて説明します。各ステップについては次の項で詳しく説明します。

  - Load Sheets マクロの使用 : 14-16 ページ
  - Save Sheets マクロの使用 : 14-17 ページ
  - データ・アップローダの実行 : 14-18 ページ
  - Data Pump の実行 : 14-18 ページ
- Oracle 以外の給与システムの計算結果のオラクル人事管理システムでの表示 : 14-19 ページ

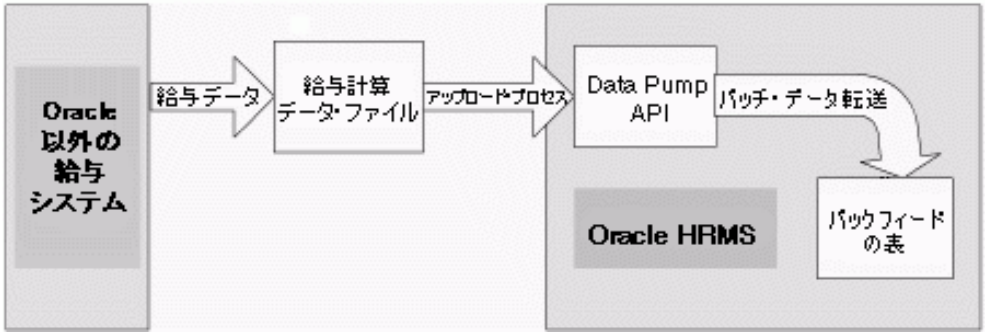
  オラクル人事管理システムのウィンドウに給与計算結果を表示する方法について説明します。
- 参照情報 : 14-21 ページ

  提供される関数、表摘要、API など、バックフィードに関連する技術情報が示されます。


# 概要

Oracle 以外の給与システムを使用する場合、Oracle Generic Third Party Payroll Backfeed により、その給与システムで提供される情報をオラクル人事管理システムの表にアップロードして、給与計算のために使用できます。この情報に、使用している Oracle 以外の給与システムで計算された支払情報およびバランス詳細を含めることができます。その後で、オラクル人事管理システムのウィンドウを使用してこの情報を表示することや、この情報に基づくレポートを作成することができます。

**図 14-1 バックフィード・プロセス**




バックフィードでアップロードした給与計算結果データは、Oracle Payroll に属する表ではなく、バックフィードの特定の表に格納されます。つまり、Oracle Payroll と Oracle 以外の給与システムを使用する場合、Oracle Payroll の実装はバックフィードによる影響を受けません。

この一般バージョンの Oracle Third Party Payroll Backfeed はベンダー独立です。実装時に、使用する Oracle 以外の給与システムおよびオラクル人事管理システム導入の要件に適した構成を行うことができます。

# Oracle Generic Third Party Payroll Backfeed の設定

一般的な給与システムのバックフィードを設定するには、次の作業手順に従います。

1. 一般的な給与システムのバックフィードのインストール

   **関連項目：** Oracle Generic Third Party Payroll Backfeed のインストール : 14-4 ページ

2. バックフィードを使用して支払情報をアップロードする場合、支払情報がオラクル人事管理システム用に設定されていることを確認します。

   **関連項目：** 支払情報 : 14-6 ページ

3. 使用している Oracle 以外の給与システムからオラクル人事管理システムにアップロードするバランス・タイプの名称を入力します。

   **関連項目：** バランス・タイプ : 14-6 ページ

4. 使用するアップロード・オプションを決定します。

   **関連項目：** 使用するアップロード・オプションの決定 : 14-10 ページ

5. Data Pump を設定します。

   **関連項目：** Data Pump の設定 : 14-8 ページ

6. Data Pump メタマッパーを実行します。

   詳細は Oracle HRMS Data Pump のテクニカル・エッセイを参照してください。

7. データ・アップローダを設定します。

   **関連項目：** データ・アップローダの設定 : 14-10 ページ

8. 「Oracle 以外の給与システムの従業員給与計算結果の表示」、「Oracle 以外の給与システムの組織実行結果の表示」および「Oracle 以外の給与システムのバランス・タイプの入力」フォーム機能をメニューに追加します。「メニュー」ウィンドウを使用します。

   **関連項目：**『Oracle Applications システム管理者ガイド』

9. 必要であれば、自社に関連する情報が表示されるように、「Oracle 以外の給与システムの従業員給与計算結果」ウィンドウおよび「Oracle 以外の給与システムの組織実行結果」ウィンドウで新規のフォルダ定義を作成します。

## Oracle Generic Third Party Payroll Backfeed のインストール

### リリース 11i.2 メンテナンス・パッチ以降

Oracle HRMS リリース 11i.2 メンテナンス・パッチ以降を使用する場合、一般的な給与システムのバックフィードはすでに Oracle Human Resources の一部としてインストール済なので、このステップはスキップできます。

### リリース 11*i*

オラクル人事管理システム 11*i* を使用する場合、次に示すパッチを適用する必要があります。これらのパッチは、Oracle サポートまたはオラクル・サポート・ホームページから入手できます。

---

**注意：** これらのパッチは変更されることがあります。最新情報については Oracle サポートにお問い合せください。

---

#### バックフィードの表のインストール : パッチ番号 1287911

このパッチにより、オラクル以外の給与システムのバックフィードの表、API、フォームおよびビューがインストールされます。

#### Data Pump 構成データのインストール : パッチ番号 1313097

このパッチにより、Data Pump からバックフィード API をコールするための Data Pump 構成データがインストールされます。Oracle Human Resources システム ID を解決する PL/SQL 関数も含まれています。これらの関数では、使用しているオラクル人事管理システムの実装に関して一定の前提条件を設定しています。これらの関数は、このドキュメントの「参照情報」の項に記載されています。使用している実装に対してその前提条件が当てはまらない場合、パッチ 1313097 で提供されるスクリプトの一部を構成する必要があります。

#### データ・アップローダのインストール : パッチ番号 1164750 および 1316578

これらのパッチにより、データ・アップローダおよびシード・データがインストールされます。それにより、ユーザーはデータ・アップローダ機能をオラクル以外の給与システムのバックフィードの一部として使用できます。パッチ 1313097 で提供される PL/SQL 関数を変更した場合、パッチ 1316578 で提供されるシード・データの変更が必要になることがあります。

#### ビジネス・ビューのインストール : パッチ番号 1315897

このパッチにより、Oracle Generic Third Party Payroll Backfeed 用のビジネス・ビューがインストールされます。このインストールが必要になるのは、たとえば、ビジネス・ビューを使用して Oracle Discoverer レポートを作成する場合です。

### リリース 11.0

オラクル人事管理システム 11.0.x を使用する場合、次に示すパッチを適用する必要があります。これらのパッチは、Oracle サポートまたはオラクル・サポート・ホームページから入手できます。

---

**注意 :** これらのパッチは変更されることがあります。最新情報については Oracle サポートにお問い合せください。

---

#### バックフィードの表のインストール : パッチ番号 1198005

このパッチにより、オラクル以外の給与システムのバックフィードの表、API、フォームおよびビューがインストールされます。

#### Data Pump のインストール : パッチ番号 1053696 および 1077660

これらのパッチにより、Data Pump の拡張機能および Data Pump からバックフィード API をコールするための Data Pump 構成データがインストールされます。Oracle Human Resources システム ID を解決する PL/SQL 関数も含まれています。これらの関数では、使用

しているオラクル人事管理システムの実装に関して一定の前提条件を設定しています。これらの関数は、このドキュメントの「参照情報」の項に記載されています。使用している実装に対してその前提条件が当てはまらない場合、パッチ 1077660 で提供されるスクリプトの一部を構成する必要があります。

**データ・アップローダのインストール : パッチ番号 1325570 および 1176584**

これらのパッチにより、データ・アップローダおよびシード・データがインストールされます。それにより、ユーザーはデータ・アップローダ機能をオラクル以外の給与システムのバックフィードの一部として使用できます。パッチ 1077660 で提供される PL/SQL 関数を変更した場合、パッチ 1176584 で提供されるシード・データの変更が必要になることがあります。

**ビジネス・ビューのインストール : パッチ番号 1198041**

このパッチにより、Oracle Generic Third Party Payroll Backfeed 用のビジネス・ビューがインストールされます。このインストールが必要になるのは、たとえば、ビジネス・ビューを使用して Oracle Discoverer レポートを作成する場合です。

## 支払情報

バックフィードを実行する前に、バックフィードを使用して支払情報をロードするすべての従業員について、オラクル人事管理システムで個人支払方法を設定する必要があります。

この情報の入力には、「組織の支払方法」ウィンドウおよび「個人支払方法」ウィンドウを使用してください。

> **関連項目：** 給与とその他の雇用グループ、および雇用情報、『Oracle HRMS ユーザーズ・ガイド』

支払詳細をアップロードするとき、通貨コードを指定する必要があります。この通貨コードは、支払方法の通貨と同じにする必要があります。

## バランス・タイプ

使用している Oracle 以外の給与システムにより保守するバランスをバックフィードの表にロードできます。バックフィードを実行する前に、バックフィードの表に格納する Oracle 以外の給与システムの各バランスを、オラクル人事管理システムにおけるバックフィード・バランス・タイプとして定義する必要があります。

---

**注意：** バックフィード・バランス・タイプと Oracle Payroll バランス・タイプとは異なります。

---

ユーザーが作成する任意のバランス・タイプに対してバランス累計期間を設定できます。各バランス・タイプに対して設定できるバランス累計期間は次のとおりです。

- 年累計バランス
- 会計年度累計バランス
- 期間累計バランス
- 月累計バランス
- 四半期累計バランス
- 計算金額

給与計算データを HRMS システムにアップロードする前に、自社で必要とされるバランス・タイプを設定する必要があります。バランス・タイプを設定するとき、それらのバランス・タイプを任意のユーザー定義エレメント入力パラメータにリンクさせることができます。それにより、バランス・タイプをそれに関連付けられたエレメントとリンクさせるレポートが簡単に作成できます。

通貨バランス額をアップロードするとき、通貨コードを指定する必要があります。この通貨コードは、必要に応じて、バランスの通貨またはバランスに関連付けられたエレメントの通貨と同じにする必要があります。ロードするバランス詳細の通貨がバランス・タイプに対して定義された通貨と同じであることを確認するために、次のいずれかのチェックが行われます。

- アップロードする金額のバランス・タイプがエレメントに関連付けられている場合、アップロードする金額が、それと関連付けられたエレメントの入力通貨と同じ通貨で表されていることを確認するチェックが行われます。
- ロードする金額のバランス・タイプがエレメントに関連付けられていない場合、アップロードする金額が、バランス・タイプに対して入力された通貨と同じ通貨で表されていることを確認するチェックが行われます。

「Oracle 以外の給与システムのバランス・タイプ」ウィンドウを使用してバランス・タイプを設定する必要があります。

## バランス・タイプを設定するには

1. バランス・タイプの表示名を入力し、有効開始日を入力します。必要であれば、有効終了日も入力できます。この日が経過すると、そのバランス・タイプは使用できなくなります。
2. 内部名を入力します。内部名は、バランス・タイプを内部で識別するために使用され、ビジネス・グループ内で一意である必要があります。
3. 必要であれば、カテゴリを入力します。カテゴリは、レポート目的でバランス・タイプをグループ化するために使用できます。たとえば、休日というカテゴリの従業員休日に関連するすべてのバランス・タイプをグループ化できます。

4. 次のいずれかを行います。

- ユーザー定義エレメント、およびバランス・タイプにリンクさせる入力パラメータを選択します。選択したエレメントおよび入力パラメータに応じて「通貨」フィールドおよび「単位」フィールドにデータが設定されます。
- バランス・タイプの単位と、必要であれば、通貨を選択します。

5. バランス・タイプにバランス額が記録されているかどうかが「使用中」チェック・ボックスに示されます。バランス額が記録されている場合、そのバランス・タイプの通貨、単位エレメント名または内部名を変更することはできません。

6. 変更を保存します。

## API

バックフィードの表のデータの保守は、ビジネス・プロセス API を使用して行います。これらの API は、Oracle の表の情報の作成、更新および削除を行うためのインタフェースです。これらの API は、1 つ以上の行ハンドラをコールします。行ハンドラは、引き渡されたデータの作成、更新または削除を行う前にそのデータを検証することにより、単一の表の中のデータを保守します。行ハンドラを直接コールしないようにしてください。

詳細はオラクル人事管理システムの API のテクニカル・エッセイを参照してください。

Data Pump を使用して、Oracle 以外の給与システムの給与計算データを Oracle HRMS バックフィードの表にアップロードすることをお薦めします。「レポートと処理の実行」ウィンドウから Data Pump をコンカレント・プログラムとして起動します。Data Pump は自動的に適切なバックフィード API をコールします。

## Data Pump の設定

Data Pump の機能の 1 つに、引き渡された他の情報を使用して内部 ID 値を解決する機能があります。Data Pump フロント・エンドを実装してこれらの ID を解決するとき、関数を作成する必要があります。各企業はオラクル人事管理システムの実装方法に応じて別々の方法でデータのマッピングを行っているので、これらの関数は実装ごとに異なります。

Data Pump を構成する前に、Oracle HRMS Data Pump のテクニカル・エッセイを参照してください。

### Data Pump フロント・エンドの構成

一般的な給与システムのバックフィードは、PER_BF_GEN_DATA_PUMP というパッケージを使用します。このパッケージには、payroll_id（この ID に対する関数は get_payroll_id と呼ばれます）などの内部システム ID を解決するために使用される関数が含まれています。

この関数定義は、pebgendp.pkh および pebgendp.pkb の 2 つのスクリプトで与えられます。オラクル人事管理システム 11.0 を使用している場合、これらのスクリプトは $PER_

TOP/patch/110/sql にあります。オラクル人事管理システム 11*i* を使用している場合、これらのスクリプトは $PER_TOP/patch/115/sql にあります。

**関連項目：** バックフィード用の Data Pump 関数 : 14-21 ページ

提供された関数で設定されている前提条件が自社に当てはまらない場合、オラクル人事管理システムの実装方法が反映されるようにそれらの関数を修正する必要があります。パッケージのコピーを作成し、そのコピーに対して変更を行うことをお薦めします。

一般関数のパラメータを変更する必要はないが、関数本体を変更する必要がある場合、この変更を行い、データベースに反して修正バージョンを実行することができます。そのためには、自分で構成したスクリプトが格納されているディレクトリに移動し、次の入力を行う必要があります。

```
sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_body_name.pkb>
```

ただし、パッケージの変更とともに、関数のパラメータの変更または新しい関数の追加が必要な場合、両方のスクリプトをデータベースに反して実行する必要があります。そのためには、自分で構成したスクリプトが格納されているディレクトリに移動し、次の入力を行う必要があります。

```
sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_header_name.pkh>
sqlplus <apps_username>/<apps_pwd>@<database_name> @<package_body_name.pkb>
```

Data Pump メタマッパーも実行する必要があります。それにより、Data Pump API およびオラクル以外の給与システムのバックフィード・インタフェースに固有のビューが再生成されます。それを行う方法および使用するその他の Data Pump 機能の詳細は、Oracle HRMS Data Pump のテクニカル・エッセイを参照してください。

提供された一般関数のパラメータの変更または新しい関数の追加を行わない場合、データ・アップローダ・フロント・エンドの構成も行う必要があります。

**関連項目：** データ・アップローダ・フロント・エンドの構成 : 14-13 ページ

# 使用するアップロード・オプションの決定

Data Pump を使用して Oracle 以外の給与システムの給与計算データをバックフィードの表にアップロードするには、最初にこのデータを Data Pump のバッチ表に挿入する必要があります。そのためには次の 2 つの方法があります。

- Data Pump メタマッパーにより生成された API の使用

  このオプションを使用する場合、給与計算結果データを読み取り、そのデータを Data Pump API を使用して Data Pump のバッチ表に挿入する PL/SQL プログラムを作成する必要があります。

- データ・アップローダの使用

  このオプションを使用する場合、データ・アップローダで読取り可能な形式に給与計算結果データをフォーマットし、フラット・ファイルに格納する必要あります。

ユーザーは、技術リソースおよび給与計算結果データのソースに基づいて、どちらが最適な方法であるかを決める必要があります。

# データ・アップローダの設定

データ・アップローダは、タブ区切りテキスト・ファイルに格納されているデータを取り出し、Data Pump メタマッパーの実行時に作成されたパッケージおよびビューを使用して、そのデータを Data Pump のバッチ表にアップロードします。データ・アップローダを使用するには、データ・アップローダに必要なフォーマットのタブ区切りファイルに給与計算結果データを挿入する必要があります。給与計算結果ファイルをフォーマットする作業を支援するために、bfexampl.xls という Microsoft Excel ワークブックが提供されています。このワークブックには、データを設定する方法が説明されています。フォーマットが終了すると、Save Sheets マクロを使用して、Excel のワークシート内のデータを、データ・アップローダで使用されるタブ区切りファイルにエクスポートできます。このマクロと Load Sheets マクロは、bfmacros.xls ファイルで提供されます。

## Excel を使用したファイルの作成

バックフィード実装の初期段階では、Excel マクロを使用して、データ・アップローダで読取り可能なファイルを作成できますが、バックフィードを本番環境で使用する場合には Excel の使用を停止してください。そのかわりに、タブ区切りのデータ・アップローダ・ファイルの作成を自動化することをお薦めします。

バックフィードのデータ・アップローダ部分の実行中に問題が発生した場合、ファイルのサイズが Excel で十分処理できるほど小さければ、デバッグのために Excel を引き続き使用できます。

## 見本ファイル

見本ファイルには次のものがあります。

### bfexampl.xls

- **ヘッダー・シート。**このシートには、個々のワークシート名など、ワークブックの基本情報が含まれています。
- **給与計算シート。**このシートには、処理日など、給与計算全般に関する詳細情報が格納されています。この情報には、create_payroll_run API により使用されるデータが含まれます。
- **バランス額シート** このシートには、給与計算ワークシートで定義されている給与計算のための従業員バランス詳細が格納されています。この情報には、create_balance_amount AP により使用されるデータが含まれています。
- **支払詳細シート**このシートには、給与計算ワークシートで定義されている給与計算のための従業員支払詳細が格納されています。この情報には、create_payment_details API により使用されるデータが含まれています。

### bfmacros.xls

- **Save Sheets マクロ。**これは、ワークブック内の個々のシートを個別のタブ区切りテキスト・ファイルとして保存するマクロです。
- **Load Sheets マクロ。**これは、ヘッダー・ファイルに基づいてテキスト・ファイルを個別にロードするマクロです。

### ヘッダー・シート。

ヘッダー・シートには、アップロードするデータのセット全体に関する情報が含まれています。ヘッダー・シートでは、バッチ名や日付などの標準情報が定義され、このアップロードで使用されるファイルも指定されます。

このアップロードを一意に識別するバッチ名を入力する必要があります。Data Pump 処理を実行すると、このバッチ名が問合せされます。

Files Start 行と Files End 行との間のテキストは、個々のシートに対するファイル名です。最初の列にはシートの名前、2 番目の列にはテキスト・ファイルの名前が入っています。これらのマクロを使用する場合に、関連するシートがこの名前のファイルとして保存されるか、またはこの名前のファイルからアップロードされます。

### 給与計算シート

すべての給与計算には、処理日、期間の開始日と終了日、および給与計算の一意の識別子に関連する情報が与えられています。このワークシートには、このタイプの情報が含まれています。

先頭のシートの Descriptor Start 列と Descriptor End 列との間に、その給与計算に関連する詳細情報があります。多くの場合、これらの情報は、ユーザーが行うすべてのデータ・アップロードに対して同じになっています。

ユーザー・キー行には、他のシートから挿入される給与計算を一意に参照するためにデータ・アップローダおよび Data Pump の機能が使用するエントリが入っています。このような参照が必要になるシートには、バランス額シートや支払詳細シートなどがあります。この参照のためのデフォルト・エントリは、%$Business Group%:payroll_identifier です。ビジネス・グループ ID と給与識別子との組合せにより常に給与計算を一意に識別する必要があるので、このエントリを変更する必要が生じないようにしてください。

ID 列には、データ・アップローダがスプレッド・シート内の行を識別する方法が示されています。同じワークシート内の他のシートがこの列を使用して、特定の行を参照できます。この場合、バランス額シートと支払詳細シートの両方に、このシート内の行を参照する payroll_run_id という列が含まれています。データの各行に異なる連番を付けて ID 列に入れる必要があります。

### バランス額シート

バランス額ワークシートには、特定の給与計算に対する各従業員に関連するバランス情報が格納されています。

Data Start 行の下の行には、その API の列タイトルが入っています。各従業員に対する給与計算バランス額詳細は、この行と Data End 行との間に入れる必要があります。すべての通貨額に対して通貨コードを指定する必要があります。

ID 列には 1 から始まる連番を入れる必要があります。

payroll_run_id という名前の列は、給与計算ワークシートの ID 列を参照します。この番号は、給与計算内のすべての行に対して同じになります。

### 支払詳細シート

支払金額シートには、給与計算で処理される各従業員の支払詳細が格納されています。

Data Start 行の下の行には、その API の列タイトルが入っています。特定の給与計算に対する支払詳細は、この行と Data End 行との間に入っている必要があります。すべての通貨額に対して通貨コードを指定する必要があります。

ID 列には 1 から始まる連番を入れる必要があります。

payroll_run_id という名前の列は、給与計算ワークシートの ID 列を参照します。この番号は、給与計算内のすべての行に対して同じになります。

### Save Sheets マクロ

この Excel マクロにより、個々のワークシートがタブ区切りテキスト・ファイルに保存されます。ヘッダー・シート以外の各テキスト・ファイルの名前はヘッダー・シートに格納されています。このマクロを実行すると、ヘッダー・シートの名前の入力を要求するプロンプトが表示されます。

### Load Sheets マクロ

このマクロを使用するには、ヘッダー・シートが格納されたタブ区切りテキスト・ファイルが必要です。このマクロにより、ヘッダー・シートで指定されたテキスト・ファイルが、マクロの実行元のワークブックにワークシートとしてロードされます。ロードするテキスト・ファイルは、選択したヘッダー・シートのテキスト・ファイルと同じディレクトリにある必要があります。

## データ・アップローダ用のアップロード・ディレクトリの指定

データ・アップローダを使用してインポートするファイルをどの場所に入れるかを指定する必要があります。そのための作業に必要なステップを次に説明します。

1. データベース管理者は、データベースの初期化ファイルで、アップロードするファイルを格納するディレクトリを指定する必要があります。そのためには、必要なディレクトリのパスを UTL_FILE_DIR パラメータで指定します。
2. システム管理者は、このディレクトリへのフル・パスを「HR: データ・エクスチェンジ・ディレクトリ」ユーザー・プロファイル・オプションで入力する必要があります。「システム・プロファイル値」ウィンドウを使用します。このプロファイル・オプションは、ユーザーが設定するセキュリティに応じて、サイト、アプリケーションおよび職責のレベルで使用できます。

## データ・アップローダ・フロント・エンドの構成

データ・アップローダの一般パラメータは、pedugens.sql という名前のスクリプトで定義されます。このスクリプトは、給与計算、バランス額などのパラメータを作成する様々な部分に分かれています。

オラクル人事管理システム 11.0 を使用している場合、このスクリプトは $PER_TOP/patch/110/sql にあります。オラクル人事管理システム 11*i* を使用している場合、このスクリプトは $PER_TOP/patch/115/sql にあります。

システム ID を解決するために Data Pump 関数のパラメータを変更したか、または Data Pump API を再生成するために新しい関数を追加し、メタマッパーを使用した場合、新しいパラメータで指定したデータが含まれている列を、Excel アップロード・ワークブックの該当のシートに格納する必要があります。参照 : アップロード・ワークブックの作成 : 14-15 ページ

次に、pedugens.sql スクリプトを修正して、Excel の列内の新しいデータを、データ・アップローダにより使用される API にマッピングする必要があります。

create_balance_amount API の詳細をマッピングするデータ・アップローダを作成するためのコード例を次に示します。

```
HRDU_DO_API_TRANSLATE.hrdu_insert_mapping(
          p_api_module        => 'create_balance_amount',
          p_column_name       => 'balance_type_name',
          p_mapped_to_name    => 'p_balance_type_name');
```

p_api_module パラメータは、この API を使用してアップロードするデータが格納されている Microsoft Excel ワークシートを識別します。この例の場合、そのワークシートは create_balance_amount です。p_column_name パラメータは、関連付けられている Excel ワークシートの列名を引き渡します。この例の場合は、balance_type_name です。p_mapped_to_name パラメータは、Excel ワークシートに関連付ける Data Pump ビュー列を引き渡します。この例の場合は、p_balance_type_name です。

アップロード・ワークブックに追加した新規の列に対する挿入文が既存または新規の関数に含まれているかどうかにかかわらず、それらの挿入文を追加する必要があります。

# バックフィードを使用して給与計算結果をアップロードする方法

バックフィードを使用して給与計算結果をアップロードするには、次の作業手順に従います。

1. 使用している Oracle 以外の給与システムからテキスト・ファイルに給与計算結果を保存します。
2. アップロード・ワークブックを作成します。

   **関連項目：** アップロード・ワークブックの作成 : 14-15 ページ

3. 給与計算データをデータ・アップローダで必要な形式にフォーマットします。

   **関連項目：** データ・アップローダで必要な形式への給与計算データのフォーマット : 14-16 ページ

4. データ・アップローダのコンカレント・プロセスを使用して、テキスト・ファイルから Data Pump バッチ表に情報をロードします。

   **関連項目：** データ・アップローダの実行 : 14-18 ページ

**注意：** データ・アップローダを使用して給与計算データを Data Pump バッチ表にロードしないが、Data Pump メタマッパーにより生成された API を使用してそのデータをアップロードする PL/SQL プログラムを作成する場合は、ステップ 3 および 4 を無視してください。

5. Data Pump コンカレント・プロセスを実行して、データを Data Pump のバッチ表からバックフィードの表にアップロードします。

**関連項目：** Data Pump の実行 : 14-18 ページ

## アップロード・ワークブックの作成

データ・アップローダを使用する前に、bfexampl.xls ファイルに基づいて、自社のニーズに合わせたアップロード・ワークブックを作成する必要があります。

ヘッダー・シートで指定されているファイルの名前を変更できます。ファイル名を変更すると、そのファイルはその名前で保存されます。たとえば、バックフィードの表にアップロードしたすべての給与計算をファイルに記録する場合、給与計算の給与識別子をそれに対応するファイルの名前の先頭に付けるのが望ましいことがあります。

後で詳細に説明するように、ワークシートのレイアウトの修正や、不要なワークシートの削除ができます。

バランス詳細機能のみを使用し、支払詳細機能は使用しない場合、支払詳細シートの詳細を示すヘッダー・シートから明細を削除することや、支払詳細シートを削除することができます。バランス詳細機能を使用しない場合も、同じ方法でその機能を削除することもできます。

check_type や ftd_amount のように、重要性が低くて使用されていない列がある場合、ワークシートからそれらの列を削除できます。データが Data Pump 経由でロードされるのを防止する列は削除しないようにしてください。たとえば、ID 列つまり給与識別子の列は Data Pump およびデータ・アップローダの動作に必須なので、削除することはできません。

それと同様に、データ列（ID 列は除く）の順序を自分の好みにあわせて変更することができます。また、Data Pump フロント・エンドに対して変更を行ったために新しい列が必要になった場合、その列を追加する必要があります。

**関連項目：** Data Pump フロント・エンドの構成 : 14-8 ページ

### データ・アップローダで必要な形式への給与計算データのフォーマット

データ・アップローダで必要な形式に給与計算データをフォーマットするには、多くの方法があります。ユーザーは自社の業務慣例に適した方法を選択できます。

1 つの方法は、オペレーティング・システムのツールを使用して給与計算データをフォーマットし、それを別のスプレッドシートにロードすることです。そして、そのデータをアップロード・ワークブック内の位置にカット・アンド・ペーストしてから、Save Sheets マクロを使用してそのワークシートを個別のタブ区切りテキスト・ファイルに保存することができます。

あるいは、Save Sheets マクロを使用して、データのないアップロード・ワークシートを保存し、オペレーティング・システムのツールを使用してデータを適切な位置に入れることができます。データが正しくフォーマットされたことを確認するには、Load Sheets マクロを使用してデータを Excel にリロードし、そのデータを表示します。Excel にデータをリロードして確認することは、データ・アップローダが正しく動作するために必須の作業ではありませんが、確認しておくことをお薦めします。

ワークシートに入っているデータがわずかである場合は、データを手動で Excel に入力してから、Save Sheets マクロを使用してそのデータを保存するという方法もあります。

## Load Sheets マクロの使用

Load Sheets マクロを使用することで、タブ区切りテキスト・ファイル・バージョンのヘッダー・シートで指定されているテキスト・ファイルをワークブックにロードできます。これらのファイルは、ヘッダー・テキスト・ファイルが格納されているディレクトリからロードされます。

### Load Sheets マクロを実行するには

1. ヘッダー・シートのバージョンが、bfexampl.xls の最初のシートと同じフォーマットで、タブ区切りテキスト・ファイルとして保存されていることを確認します。このファイルでは、ロードするテキスト・ファイルと、それらのファイルをロードするときに作成する必要がある Excel ワークシートの名前が定義されています。
2. アップロードするテキスト・ファイルが、ヘッダー・シート・テキスト・ファイルと同じディレクトリに格納されていることを確認します。
3. ファイルのロード先のワークブックを開きます。このワークブックに Load Sheets マクロが含まれていない場合、別のワークブックからそのマクロをコピーする必要があります。
4. 「ツール」メニューで「マクロ」を選択し、表示された「マクロ」ウィンドウで Load Sheets マクロを選択します。
5. ヘッダー・シート・テキスト・ファイルが格納されているディレクトリのパスを入力し、「OK」を選択します。

**注意：** このパスの最後に文字 "¥" を入力する必要があります。たとえば、C:¥upload¥ と入力します。

6. ヘッダー・シート・テキスト・ファイルの名前を入力し、「OK」を選択します。それらのファイルがワークブックにロードされます。

**注意：** ファイルがワークブックにロードされると、ヘッダー情報が入っているワークシート（つまり、最初のワークシート）の名前は必ず header_sheet になります。

## Save Sheets マクロの使用

Save Sheets マクロを使用することで、複数シートの Excel ワークブックを対応するタブ区切りテキスト・ファイルに保存できます。各テキスト・ファイルは、ヘッダー・シートで指定された名前が付けられ、指定されたディレクトリに保存されます。ワークブックの最初のシートであるヘッダー・シートは、データ・アップローダにより使用されるヘッダー・ファイルを作成します。

### Save Sheets マクロを実行するには

1. 必要な Excel ワークブックが開いていることを確認します。このワークブックに Save Sheets マクロが含まれていない場合、別のワークブックからそのマクロをコピーする必要があります。
2. ヘッダー情報が含まれているワークブックが header_sheet という名前になっていることを確認します。その名前になっていないと、このワークシートまたはマクロは失敗します。
3. 「ツール」メニューで「マクロ」を選択し、表示された「マクロ」ウィンドウで Save Sheets マクロを選択します。
4. テキスト・ファイルを保存するディレクトリのパスを入力します。このディレクトリとして、バックフィードの設定時にシステム管理者が定義したディレクトリを指定する必要があります。「OK」を選択します。

**注意：** このパスの最後に文字 "¥" を入力する必要があります。たとえば、C:¥upload¥ と入力します。

5. ヘッダー・ファイルの名前を入力します。デフォルトでは、この名前は、ワークブックの最初のワークシートの名前になります。データ・アップローダ・プロセスを実行するとき、このファイルを指定する必要があります。「OK」を選択します。

## データ・アップローダの実行

データ・アップローダは、ユーザーが作成したテキスト・ファイルに格納されている情報を取り出し、その情報を Data Pump のバッチ表にロードします。各アップロードで使用されるファイルは、HR データ・アップローダのコンカレント・プロセスの実行時にユーザーが選択するヘッダー・ファイルにより定義されます。

**注意：** 必要であれば、別の方法を使用して、給与計算データを Data Pump の表にロードできます。

設定タスクが終了した後、「要求の発行」ウィンドウでデータ・アップローダを実行します。

### データ・アップローダ・プロセスを実行するには

1. アップロードするファイルが、バックフィードの設定時にデータベース管理者およびシステム管理者が指定したディレクトリに格納されていることを確認します。
2. 「要求の発行」ウィンドウで、HR データ・アップローダのコンカレント・プロセスを選択します。
3. 使用するヘッダー・ファイルのファイル名を入力し、「発行」を選択します。

### データ・アップローダを使用したエラー追跡

データ・アップローダの使用中にエラーが発生した場合、エラーの詳細を確認するためにコンカレント要求ログ・ファイルを表示する必要があります。

## Data Pump の実行

給与計算データを Data Pump のバッチ表に入れた後、Data Pump エンジンのコンカレント・プロセスを実行して、そのデータをバックフィードの表にアップロードする必要があります。

### Data Pump エンジンのコンカレント・プロセスを実行するには

1. Data Pump エンジンのコンカレント・プロセスを選択します。
2. 必要なバッチ名を入力し、プロセスを検証するかどうかを指定します。

   バッチ名の形式は、< バッチ名 >-< バッチ ID> となります。バッチ名はヘッダー・ファイルに入力されたバッチ名に対応しており、バッチ ID は内部的に割り当てられる ID です。たとえば、次のようになります。

   Week12-1234

3. 「発行」を選択します。

Data Pump で発生したエラーの検出とその訂正については、Oracle HRMS Data Pump のテクニカル・エッセイを参照してください。

# Oracle 以外の給与システムの計算結果のオラクル人事管理システムでの表示

Oracle 以外の給与システムの給与計算結果をバックフィードの表にアップロードした後、その計算結果を次の方法で表示できます。

- 従業員ごとの表示（「Oracle 以外の給与システムの従業員実行結果」ウィンドウ）
- 組織、役職、等級、グループ、職階または事業所ごとの表示（「Oracle 以外の給与システムの組織実行結果」ウィンドウ）

これらのウィンドウにはそれぞれ、「バランス詳細」および「支払詳細」の 2 つのフォルダがあります。それにより、ユーザーは、標準的なフォルダ・ユーティリティを使用して必要な情報を表示できます。

### 次の手順で「Oracle 以外の給与システムの従業員実行結果」ウィンドウを使用して給与計算結果詳細を問い合せます。

1. 次のうちの 1 つ以上を実行します。
   - 個人名に関して氏名または部分名の問合せを入力します。個人に対してプリフィクスが定義されている場合、氏名の問合せは 'Maddox, Miss Julie' というフォーマットで入力する必要があります。
   - 従業員番号、アサイメント番号、給与または給与識別子の問合せを入力します。
   - 給与期間開始日および終了日の最早日および最遅日、および給与処理日を指定します。これで、給与計算結果の範囲を検索できます。
2. 「検索」ボタンを選択します。

   問合せにより検索された給与計算結果は「Oracle 以外の給与システムの従業員実行結果」ウィンドウに表示されます。問合せにより複数のレコードが検索された場合、[ ↓ ] キーを使用するか、または「進む」メニューで「次のレコード」を選択して、次のレコードを表示します。

### 次の手順で「Oracle 以外の給与システムの組織実行結果」ウィンドウを使用して給与計算結果詳細を問い合せます。

1. 次のうちの 1 つ以上を実行します。
   - 組織、People グループ、役職、職階、等級または事業所の問合せを入力します。
   - 給与または給与識別子の問合せを入力します。
   - 給与期間開始日および終了日の最早日および最遅日、および給与処理日を指定します。これで、給与計算結果の範囲を検索できます。
2. 「検索」ボタンを選択します。

   問合せにより検索された給与計算結果は「Oracle 以外の給与システムの組織実行結果」ウィンドウに表示されます。問合せにより複数のレコードが検索された場合、[ ↓ ] キーを使用するか、または「進む」メニューで「次のレコード」を選択して、次のレコードを表示します。

### 次の手順で Oracle 以外の給与システムの給与計算結果を表示します。

1. 「Oracle 以外の給与システムの従業員実行結果の検索」ウィンドウまたは「Oracle 以外の給与システムの組織実行結果の検索」ウィンドウを使用して、必要な情報を検索します。
   - 「Oracle 以外の給与システムの従業員実行結果の検索」ウィンドウを使用して問合せを行った場合、従業員および給与計算の詳細が表示されます。
   - 「Oracle 以外の給与システムの組織実行結果の検索」ウィンドウを使用して問合せを行った場合、給与計算の詳細が表示されます。検索ウィンドウはバックグラウンドで開いたままになっているので、検索ウィンドウを参照することにより、表示されている結果を検索した問合せを確認できます。
2. 「バランス詳細」代替リージョンを選択します。このリージョンには、金額、会計年度累計額、エレメント名など、表示されている従業員および給与計算に関連するすべてのバランス情報が表示されます。標準的なフォルダ・ツールを使用して、このフォルダに表示されているデータを制御できます。
3. 「支払詳細」代替リージョンを選択します。このリージョンには、小切手番号、支払日、金額など、表示されている従業員および給与計算に関連するすべての支払情報が表示されます。標準的なフォルダ・ツールを使用して、このフォルダに表示されているデータを制御できます。

# 参照情報

## バックフィード用の Data Pump 関数

次の表で、PER_BF_GEN_DATA_PUMP パッケージに含まれている関数を説明します。これらの関数は、バックフィードの表に対するオラクル人事管理システムのシステム ID を取得するために使用されます。これらの関数では、バックフィード経由でアップロードされるデータついて前提条件を設定しています。これらの前提条件が自社に当てはまらない場合、1 つ以上の関数の構成が必要になることがあります。

関数の定義は、pebgendp.pkh および pebgendp.pkb の 2 つのスクリプトで提供されます。オラクル人事管理システム 11.0 を使用している場合、これらのスクリプトは $PER_TOP/patch/110/sql にあります。オラクル人事管理システム 11*i* を使用している場合、これらのスクリプトは $PER_TOP/patch/115/sql にあります。

Data Pump フロント・エンドの構成 : 14-8 ページを参照してください。

| 関数名 | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| GET_PAYROLL_RUN_ID | 給与計算識別子およびビジネス・グループ ID を指定されて、payroll_run_id を戻します。 | | | |
| | P_PAYROLL_RUN_USER_KEY | VARCHAR2 | IN | HR_PUMP_BATCH_LINE_USER_KEYS の主キー |
| GET_BALANCE_TYPE_ID | バランス・タイプ名、ビジネス・グループ ID および有効日を指定されて、balance_type_id を戻します。 | | | |
| | P_BALANCE_TYPE_NAME | VARCHAR2 | IN | バランス・タイプの名前 |
| | P_BUSINESS_GROUP_ID | NUMBER | IN | ビジネス・グループ ID。ビジネス・グループ名がバッチ・ヘッダー表にアップロードされている場合、この ID は Data Pump によって決定されます。 |
| | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| GET_PAYROLL_ID | 給与、ビジネス・グループ ID および有効日を指定されて、payroll_id を戻します。 | | | |
| | P_PAYROLL_NAME | VARCHAR2 | IN | 給与の名称 |
| | P_BUSINESS_GROUP_ID | NUMBER | IN | ビジネス・グループ ID。ビジネス・グループ名がバッチ・ヘッダー表にアップロードされている場合、この ID は Data Pump によって決定されます。 |

| 関数名 | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| GET_ASSIGNMENT_ID | 従業員番号、ビジネス・グループ ID および有効日を指定されて、プライマリ・アサイメントに対する assignment_id を戻します。 | | | |
| | P_EMPLOYEE_NUMBER | VARCHAR2 | IN | 従業員の番号 |
| | P_BUSINESS_GROUP_ID | NUMBER | IN | per_business_groups への FK |
| | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| GET_PERSONAL_PAYMENT_METHOD_ID | 従業員番号、ビジネス・グループ ID、有効日および支払方法名を指定されて、プライマリ・アサインメントに対する personal_payment_method_id を戻します。 | | | |
| | P_EMPLOYEE_NUMBER | VARCHAR2 | IN | 従業員の一意番号 |
| | P_BUSINESS_GROUP_ID | NUMBER | IN | per_business_groups への FK |
| | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| | P_ORG_PAYMENT_METHOD_NAME | VARCHAR2 | IN | 組織支払方法名 |
| GET_INPUT_VALUE_ID | エレメントのレポート名、ビジネス・グループ ID および有効日を指定されて、エレメントの最初の入力パラメータ（結果値ではない）に対する input_value_id を戻します。 | | | |
| | P_REPORTING_NAME | VARCHAR2 | IN | ビジネス・グループ内で一意。エレメントのレポート名。 |
| | P_BUSINESS_GROUP_ID | NUMBER | IN | PER_BUSINESS_GROUPS への FK |
| | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |

# 表の説明

次の項では、バックフィードで使用される特定の表についてさらに詳しく説明します。これらの表は、バックフィードの一部として提供されます。これらの表はオラクル人事管理システムのリリース 11.0 には含まれていません。

## PER_BF_BALANCE_AMOUNTS

PER_BF_BALANCE_AMOUNTS には、バックフィード・バランスのアサイメント・レベル計算額および要約値が給与処理期間ごとに格納されています。

外部キー

| 主キー表 | 主キー列 | 外部キー列 |
|---|---|---|
| HR_ALL_ORGANIZATION_UNITS | ORGANIZATION_ID | BUSINESS_GROUP_ID |
| PER_BF_BALANCE_TYPES | BALANCE_TYPE_ID | BALANCE_TYPE_ID |
| PER_BF_PROCESSED_ASSIGNMENTS | PROCESSED_ASSIGNMENT_ID | PROCESSED_ASSIGNMENT_ID |

列の説明

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| BALANCE_AMOUNT_ID (PK) | NOT NULL | NUMBER(9) | 一意のシステム識別子 |
| BALANCE_TYPE_ID | NOT NULL | NUMBER(9) | PER_BF_BALANCE_TYPES への FK |
| PROCESSED_ASSIGNMENT_ID | NOT NULL | NUMBER(9) | PER_BF_PROCESSED_ASSIGNMENTS への FK |
| BUSINESS_GROUP_ID | NOT NULL | NUMBER(15) | HR_ALL_ORGANIZATION_UNITS への FK |
| YTD_AMOUNT | NULL | NUMBER(15) | 年累計バランス額 |
| FYTD_AMOUNT | NULL | NUMBER(15) | 会計年度累計バランス額 |
| PTD_AMOUNT | NULL | NUMBER(15) | 期間累計バランス額 |
| MTD_AMOUNT | NULL | NUMBER(15) | 月累計バランス額 |
| QTD_AMOUNT | NULL | NUMBER(15) | 四半期累計バランス額 |
| RUN_AMOUNT | NULL | NUMBER(15) | 給与計算額バランス |
| OBJECT_VERSION_NUMBER | NULL | NUMBER(9) | システムが生成する行のバージョン |
| LAST_UPDATE_DATE | NULL | DATE | 標準 WHO 列 |
| LAST_UPDATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| LAST_UPDATE_LOGIN | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATION_DATE | NULL | DATE | 標準 WHO 列 |

索引

| 索引名 | 索引タイプ | 連番 | 列名 |
|---|---|---|---|
| PER_BF_BALANCE_AMOUNTS_FK2 | NOT UNIQUE | 1 | BALANCE_TYPE_ID |
| PER_BF_BALANCE_AMOUNTS_FK3 | NOT UNIQUE | 1 | PROCESSED_ASSIGNMENT_ID |
| PER_BF_BALANCE_AMOUNTS_UK1 | UNIQUE | 1 | BALANCE_TYPE_ID |
| | | 2 | PROCESSED_ASSIGNMENT_ID |

連番

| 連番 | 導出列 |
|---|---|
| PER_BF_BALANCE_AMOUNTS_S | BALANCE_AMOUNT_ID |

## PER_BF_BALANCE_TYPES

PER_BF_BALANCE_TYPES には、バックフィードの表に挿入できるバランスのタイプが格納されています。

外部キー

| 主キー表 | 主キー列 | 外部キー列 |
|---|---|---|
| HR_ALL_ORGANIZATION_UNITS | ORGANIZATION_ID | BUSINESS_GROUP_ID |

列の説明

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| BALANCE_TYPE_ID (PK) | NOT NULL | NUMBER(9) | システムが生成するこの行の一意識別子 |
| INPUT_VALUE_ID | NULL | NUMBER(9) | PAY_INPUT_VALUES_F への FK |
| BUSINESS_GROUP_ID | NOT NULL | NUMBER(15) | HR_ALL_ORGANIZATION_UNITS への FK |
| DISPLAYED_NAME | NOT NULL | VARCHAR2(30) | ユーザーの確認のために表示されるバランス・タイプ名 |
| INTERNAL_NAME | NOT NULL | VARCHAR2(30) | バランスの内部名 |
| UOM | NULL | VARCHAR2(30) | バランスの単位 |
| CURRENCY | NULL | VARCHAR2(15) | 単位が金額である場合、バランスの通貨 |
| CATEGORY | NULL | VARCHAR2(30) | レポート目的のためにバランス・タイプのグループ化が可能 |
| DATE_FROM | NULL | DATE | このタイプの有効期間の開始日 |
| DATE_TO | NULL | DATE | このタイプの有効期間の終了日 |
| OBJECT_VERSION_NUMBER | NULL | NUMBER(9) | システムが生成する行のバージョン |
| LAST_UPDATE_DATE | NULL | DATE | 標準 WHO 列 |
| LAST_UPDATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| LAST_UPDATE_LOGIN | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATION_DATE | NULL | DATE | 標準 WHO 列 |

索引

| 索引名 | 索引タイプ | 連番 | 列名 |
|---|---|---|---|
| PER_BF_BALANCE_TYPES_N1 | NOT UNIQUE | 1 | INPUT_VALUE_ID |
| PER_BF_BALANCE_TYPES_UK1 | UNIQUE | 1 | DISPLAYED_NAME |
| | | 2 | BUSINESS_GROUP_ID |
| PER_BF_BALANCE_TYPES_UK2 | UNIQUE | 1 | INTERNAL_NAME |
| | | 2 | BUSINESS_GROUP_ID |

連番

| 連番 | 導出列 |
|---|---|
| PER_BF_BALANCE_TYPES_S | BALANCE_TYPE_ID |

## PER_BF_PAYMENT_DETAILS

PER_BF_PAYMENT_DETAILS には、Oracle 以外の給与システムのプロバイダにより処理された支払詳細がアサイメントごとに格納されています。

外部キー

| 主キー表 | 主キー列 | 外部キー列 |
|---|---|---|
| HR_ALL_ORGANIZATION_UNITS | ORGANIZATION_ID | BUSINESS_GROUP_ID |
| PER_BF_PROCESSED_ASSIGNMENTS | PROCESSED_ASSIGNMENT_ID | PROCESSED_ASSIGNMENT_ID |

列の説明

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| PAYMENT_DETAIL_ID (PK) | NOT NULL | NUMBER(9) | システムが生成する行の一意識別子 |
| PROCESSED_ASSIGNMENT_ID | NOT NULL | NUMBER(9) | PER_BF_PROCESSED_ASSIGNMENTS への FK |
| PERSONAL_PAYMENT_METHOD_ID | NOT NULL | NUMBER(9) | PAY_PERSONAL_PAYMENT_METHODS_F への FK |
| BUSINESS_GROUP_ID | NOT NULL | NUMBER(15) | ビジネス・グループ ID |
| CHECK_NUMBER | NULL | NUMBER(15) | 発行された小切手の番号 |
| PAYMENT_DATE | NULL | DATE | 小切手の発行日 |
| AMOUNT | NULL | NUMBER(15) | 金額 |
| CHECK_TYPE | NULL | VARCHAR2(30) | 国固有タイプ |
| OBJECT_VERSION_NUMBER | NULL | NUMBER(9) | システムが生成する行のバージョン |
| LAST_UPDATE_DATE | NULL | DATE | 標準 WHO 列 |
| LAST_UPDATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| LAST_UPDATE_LOGIN | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATION_DATE | NULL | DATE | 標準 WHO 列 |

索引

| 索引名 | 索引タイプ | 連番 | 列名 |
|---|---|---|---|
| PER_BF_PAYMENT_DETAILS_UK1 | UNIQUE | 1 | PROCESSED_ASSIGNMENT_ID |
| | | 2 | PERSONAL_PAYMENT_METHOD_ID |

連番

| 連番 | 導出列 |
|---|---|
| PER_BF_PAYMENT_DETAILS_S | PAYMENT_DETAIL_ID |

## PER_BF_PAYROLL_RUNS

PER_BF_PAYROLL_RUNS には、Oracle 以外の給与システムのプロバイダが処理した給与計算が格納されています。

外部キー

| 主キー表 | 主キー列 | 外部キー列 |
|---|---|---|
| HR_ALL_ORGANIZATION_UNITS | ORGANIZATION_ID | BUSINESS_GROUP_ID |

列の説明

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| PAYROLL_RUN_ID (PK) | NOT NULL | NUMBER(9) | システムが生成する行の一意識別子 |
| PAYROLL_ID | NOT NULL | NUMBER(9) | PAY_ALL_PAYROLLS_F への FK |
| BUSINESS_GROUP_ID | NOT NULL | NUMBER(15) | ビジネス・グループ ID |
| PAYROLL_IDENTIFIER | NOT NULL | VARCHAR2(30) | 同じ期間内に実行された可能性のある他の給与計算からこの給与計算を一意に識別するための識別子 |
| PERIOD_START_DATE | NULL | DATE | 給与期間の開始日 |
| PERIOD_END_DATE | NULL | DATE | 給与期間の終了日 |
| PROCESSING_DATE | NULL | DATE | 給与の処理日 |
| OBJECT_VERSION_NUMBER | NULL | NUMBER(9) | システムが生成する行のバージョン。更新のたびに 1 ずつ増加。 |
| LAST_UPDATE_DATE | NULL | DATE | 標準 WHO 列 |
| LAST_UPDATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| LAST_UPDATE_LOGIN | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATION_DATE | NULL | DATE | 標準 WHO 列 |

索引

| 索引名 | 索引タイプ | 連番 | 列名 |
|---|---|---|---|
| PER_BF_PAYROLL_RUNS_N1 | NOT UNIQUE | 1 | PAYROLL_ID |
| PER_BF_PAYROLL_RUNS_UK1 | UNIQUE | 1 | PAYROLL_IDENTIFIER |
| | | 2 | BUSINESS_GROUP_ID |

連番

| 連番 | 導出列 |
|---|---|
| PER_BF_PAYROLL_RUNS_S | PAYROLL_RUN_ID |

## PER_BF_PROCESSED_ASSIGNMENTS

PER_BF_PROCESSED_ASSIGNMENT は、Oracle 以外の給与システムのプロバイダが処理したアサイメントの詳細が給与計算ごとに格納された交差部表です。

外部キー

| 主キー表 | 主キー列 | 外部キー列 |
|---|---|---|
| PER_BF_PAYROLL_RUNS | PAYROLL_RUN_ID | PAYROLL_RUN_ID |

列の説明

| 名称 | NULL? | タイプ | 説明 |
|---|---|---|---|
| PROCESSED_ASSIGNMENT_ID (PK) | NOT NULL | NUMBER(9) | システムが生成する一意識別子 |
| PAYROLL_RUN_ID | NOT NULL | NUMBER(9) | PER_BF_PAYROLL_RUN への FK |
| ASSIGNMENT_ID | NOT NULL | NUMBER(9) | PER_ALL_ASSIGNMENTS_F への FK |
| OBJECT_VERSION_NUMBER | NULL | NUMBER(9) | システムが生成する行のバージョン。更新のたびに 1 ずつ増加。 |
| LAST_UPDATE_DATE | NULL | DATE | 標準 WHO 列 |
| LAST_UPDATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| LAST_UPDATE_LOGIN | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATED_BY | NULL | NUMBER(15) | 標準 WHO 列 |
| CREATION_DATE | NULL | DATE | 標準 WHO 列 |

索引

| 索引名 | 索引タイプ | 連番 | 列名 |
|---|---|---|---|
| PER_BF_PROCESSED_ASSIGNS_N1 | NOT UNIQUE | 1 | PAYROLL_RUN_ID |
| PER_BF_PROCESSED_ASSIGNS_N2 | NOT UNIQUE | 1 | ASSIGNMENT_ID |
| PER_BF_PROCESSED_ASSIGNS_UK1 | UNIQUE | 1 | PAYROLL_RUN_ID |
| | | 2 | ASSIGNMENT_ID |

連番

| 連番 | 導出列 |
|---|---|
| PER_BF_PROCESSED_ASSIGNMENTS_S | PROCESSED_ASSIGNMENT_ID |

# Data Pump が生成するパッケージおよびビュー

Data Pump メタマッパーを実行すると、API と、作成されたユーザー・キーおよび関数に基づいて、ビューおよびパッケージが生成されます。

給与計算結果データを Data Pump のバッチ表に挿入する方法は、次の 2 つがあります。

**方法 1:**
ユーザーが自分で給与計算結果データを Data Pump のバッチ表に挿入する場合、この項で説明される Data Pump パッケージおよびビューを使用する必要があります。

**方法 2:**
データ・アップローダを使用する場合、Data Pump のバッチ表示に自動的にデータが挿入されます。

## 提供されたユーザー・キーおよび関数を使用して Data Pump メタマッパーを実行するときに生成されるパッケージ

| | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| パッケージ・プロシージャ名：**HRDPP_CREATE_PAYROLL_RUN.INSERT_BATCH_LINES** | | | | |
| このプロシージャにより、PER_BF_PAYROLL_RUNS 表の HR_PUMP_BATCH_LINES にデータが挿入されます。 | | | | |
| 必須列 | P_BATCH_ID | NUMBER | IN | HR_PUMP_BATCH_LINES からの PK |
| | P_USER_SEQUENCE | NUMBER | IN | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |
| | P_LINK_VALUE | NUMBER | IN | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| 必須列 | P_PAYROLL_IDENTIFIER | VARCHAR2 | IN | PER_BF_PAYROLL_RUNS 内の列 |
| | P_PERIOD_START_DATE | DATE | IN | PER_BF_PAYROLL_RUNS 内の列 |
| | P_PERIOD_END_DATE | DATE | IN | PER_BF_PAYROLL_RUNS 内の列 |
| | P_PROCESSING_DATE | DATE | IN | PER_BF_PAYROLL_RUNS 内の列 |

| | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| | P_PAYROLL_RUN_USER_KEY | VARCHAR2 | IN/OUT | Data Pump の表にアップロードする各行の一意識別子が格納されています。この識別子は、バックフィードの表への移送時に解決されます。user_key は、business_group_name と payroll_identifier の組合せです。ユーザー・キーは、関連するバランス詳細および支払詳細に対する外部キーとして使用されます。 |
| 必須列 | P_PAYROLL_NAME | VARCHAR2 | IN | 給与の名称。payroll_id を解決するために関数 GET_PAYROLL_ID により使用されます。 |
| 必須列 | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| パッケージ・プロシージャ名：**HRDPP_CREATE_BALANCE_AMOUNT.INSERT_BATCH_LINES**<br>このプロシージャにより、PER_BF_BALANCE_AMOUNTS 表の HR_PUMP_BATCH_LINES にデータが挿入されます。 | | | | |
| 必須列 | P_BATCH_ID | NUMBER | IN | HR_PUMP_BATCH_LINES からの PK |
| | P_USER_SEQUENCE | NUMBER | IN | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |
| | P_LINK_VALUE | NUMBER | IN | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。 並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| 必須列 | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| | P_YTD_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |
| | P_FYTD_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |
| | P_PTD_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |

| | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| | P_MTD_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |
| | P_QTD_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |
| | P_RUN_AMOUNT | NUMBER | IN | PER_BF_BALANCE_AMOUNTS 内の列 |
| 必須列 | P_BALANCE_TYPE_NAME | VARCHAR2 | IN | バランス・タイプの名称。関数 GET_BALANCE_TYPE_ID により解決される FK 参照。 |
| 必須列 | P_EMPLOYEE_NUMBER | VARCHAR2 | IN | 従業員の番号。assignment_id を解決するために関数 GET_ASSIGNMENT_ID によって使用されます。 |
| 必須列 | P_PAYROLL_RUN_USER_KEY | VARCHAR2 | IN / OUT | 代理 ID が PER_BF_PAYROLL_RUNS に挿入され外部キーとして使用可能になった後、その代理 ID を取得できるようにするために作成される ID |
| パッケージ・プロシージャ名：**HRDPP_CREATE_PAYMENT_DETAIL.INSERT_BATCH_LINES**<br>このプロシージャにより、PER_BF_PAYMENT_DETAILS 表の HR_PUMP_BATCH_LINES にデータが挿入されます。 | | | | |
| 必須列 | P_BATCH_ID | NUMBER | IN | HR_PUMP_BATCH_LINES からの PK |
| | P_USER_SEQUENCE | NUMBER | IN | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |
| | P_LINK_VALUE | NUMBER | IN | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。 並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| 必須列 | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| | P_CHECK_NUMBER | VARCHAR2 | IN | PER_BF_PAYMENT_DETAILS 内の列 |
| | P_PAYMENT_DATE | DATE | IN | PER_BF_PAYMENT_DETAILS 内の列 |

| | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| | P_AMOUNT | NUMBER | IN | PER_BF_PAYMENT_DETAILS 内の列 |
| | P_CHECK_TYPE | VARCHAR2 | IN | PER_BF_PAYMENT_DETAILS 内の列 |
| 必須列 | P_PAYROLL_RUN_USER_KEY | VARCHAR2 | IN / OUT | 代理 ID が PER_BF_PAYMENT_DETAILS に挿入され外部キーとして使用可能になった後、その代理 ID を取得できるようにするために作成される ID |
| 必須列 | P_EMPLOYEE_NUMBER | VARCHAR2 | IN | 従業員の番号。assignment_id を解決するために関数 GET_ASSIGNMENT_ID によって使用されます。 |
| 必須列 | P_PRIORITY | NUMBER | IN | 個人支払方法の優先順位 |
| 必須列 | P_ORG_PAYMENT_METHOD_NAME | VARCHAR2 | IN | 組織支払方法の名称。この名称は、PAY_PERSONAL_PAYMENT_METHODS_F の内部 ID を解決して、PER_BF_PAYMENT_DETAILS 内の外部キーとして使用可能にするするために使用されます。この ID を解決するために関数 GET_PERSONAL_PAYMENT_METHOD_ID が使用されます。 |
| パッケージ・プロシージャ名：**HRDPP_CREATE_BALANCE_TYPE.INSERT_BATCH_LINES**<br>このプロシージャにより、PER_BF_BALANCE_TYPE 表の HR_PUMP_BATCH_LINES にデータが挿入されます。 | | | | |
| 必須列 | P_BATCH_ID | NUMBER | IN | HR_PUMP_BATCH_LINES からの PK |
| | P_USER_SEQUENCE | NUMBER | IN | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |
| | P_LINK_VALUE | NUMBER | IN | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。 並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |

| | パラメータ | タイプ | IN/OUT | 説明 |
|---|---|---|---|---|
| 必須列 | P_EFFECTIVE_DATE | DATE | IN | アップロードの有効日 |
| 必須列 | P_BALANCE_TYPE_USER_KEY | VARCHAR2 | IN / OUT | 代理 ID が PER_BF_BALANCE_TYPES に挿入され外部キーとして使用可能になった後、その代理 ID を取得できるようにするために作成される ID |
| 必須列 | P_DISPLAYED_NAME | VARCHAR2 | IN | DISPLAYED_NAME 列にアップロードするデータ |
| 必須列 | P_INTERNAL_NAME | VARCHAR2 | IN | INTERNAL_NAME 列にアップロードするデータ |
| | P_UOM | VARCHAR2 | IN | UOM 列にアップロードするデータ |
| | P_CURRENCY | VARCHAR2 | IN | CURRENCY 列にアップロードするデータ |
| | P_DATE_FROM | DATE | IN | DATE_FROM 列にアップロードするデータ |
| | P_DATE_TO | DATE | IN | DATE_TO 列にアップロードするデータ |
| | P_DISPLAY_SEQUENCE | NUMBER | IN | input_value_id の解決に役立つ入力パラメータの表示順序 |
| | P_REPORTING_NAME | VARCHAR2 | IN | この ID を解決するエレメントのレポート名 |

## 提供されたユーザー・キーおよび関数を使用して Data Pump メタマッパーを実行するときに生成されるビュー

| パラメータ | 説明 |
|---|---|
| ビュー名 **HRDPV_CREATE_PAYROLL_RUN**<br>HR_PUMP_BATCH_LINES 表でのビュー。PER_BF_PAYROLL_RUNS 表にアップロードするデータが表示されます。 | |
| BATCH_ID | HR_PUMP_BATCH_HEADERS への FK |
| BATCH_LINE_ID | HR_PUMP_BATCH_LINES への FK |
| API_MODULE_ID | HR_PUMP_API_MODULES への FK |

| パラメータ | 説明 |
|---|---|
| LINE_STATUS | 明細のステータス<br>U - 未処理 - Data Pump は実行されていない<br>C - 完了 - Data Pump は正常に実行されている<br>E - エラー - Data Pump は終了したがエラーが発生した |
| USER_SEQUENCE | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |
| LINK_VALUE | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| P_EFFECTIVE_DATE | 給与計算アップロードの有効日 |
| P_PAYROLL_IDENTIFIER | PAYROLL_IDENTIFIER 列にアップロードするデータ |
| P_PERIOD_START_DATE | PERIOD_START_DATE 列にアップロードするデータ |
| P_PERIOD_END_DATE | PERIOD_END_DATE 列にアップロードするデータ |
| P_PROCESSING_DATE | PROCESSING_DATE 列にアップロードするデータ |
| P_PAYROLL_RUN_USER_KEY | 行を一意に識別するユーザー・キー。データをアップロードするとき代理 FK として使用されます。 |
| P_OBJECT_VERSION_NUMBER | オブジェクト・バージョン番号 |
| P_PAYROLL_NAME | 給与の名称 |
| ビュー名 **HRDPV_CREATE_BALANCE_AMOUNT**<br>HR_PUMP_BATCH_LINES 表でのビュー。PER_BF_BALANCE AMOUNT 表にアップロードするデータが表示されます。 | |
| BATCH_ID | HR_PUMP_BATCH_HEADERS への FK |
| BATCH_LINE_ID | HR_PUMP_BATCH_LINES への FK |
| API_MODULE_ID | HR_PUMP_API_MODULES への FK |
| LINE_STATUS | 明細のステータス<br>U - 未処理 - Data Pump は実行されていない<br>C - 完了 - Data Pump は正常に実行されている<br>E - エラー - Data Pump は終了したがエラーが発生した |
| USER_SEQUENCE | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |

| パラメータ | 説明 |
| --- | --- |
| LINK_VALUE | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| P_EFFECTIVE_DATE | 給与計算アップロードの有効日 |
| P_YTD_AMOUNT | YTD_AMOUNT 列にアップロードするデータ |
| P_FYTD_AMOUNT | FYTD_AMOUNT 列にアップロードするデータ |
| P_PTD_AMOUNT | PTD_AMOUNT 列にアップロードするデータ |
| P_MTD_AMOUNT | MTD_AMOUNT 列にアップロードするデータ |
| P_QTD_AMOUNT | QTD_AMOUNT 列にアップロードするデータ |
| P_RUN_AMOUNT | RUN_AMOUNT 列にアップロードするデータ |
| P_PROCESSED_ASSIGNMENT_ID | PER_BF_PROCESSED_ASSIGNMENTS からの PK |
| P_PROCESSED_ASSIGNMENT_OVN | PER_BF_PROCESSED_ASSIGNMENTS からの OVN |
| P_BALANCE_AMOUNT_ID | PER_BF_BALANCE_AMOUNT からの PK |
| P_BALANCE_AMOUNT_OVN | PER_BF_BALANCE_AMOUNT からの OVN |
| P_BALANCE_TYPE_NAME | バランス・タイプ名 |
| P_EMPLOYEE_ NUMBER | 従業員番号 |
| P_PAYROLL_RUN_USER_KEY | PER_BF_PAYROLL_RUNS からの代理 FK |
| ビュー名 **HRDPV_CREATE_PAYMENT_DETAIL**<br>HR_PUMP_BATCH_LINES 表でのビュー。PER_BF_PAYMENT_DETAILS 表にアップロードするデータが表示されます。 | |
| BATCH_ID | HR_PUMP_BATCH_HEADERS への FK |
| BATCH_LINE_ID | HR_PUMP_BATCH_LINES への FK |
| API_MODULE_ID | HR_PUMP_API_MODULES への FK |
| LINE_STATUS | 明細のステータス<br>U - 未処理 - Data Pump は実行されていない<br>C - 完了 - Data Pump は正常に実行されている<br>E - エラー - Data Pump は終了したがエラーが発生した |
| USER_SEQUENCE | 行を挿入する順序が重要である場合、連番で処理の順序を制御できます。 |

| パラメータ | 説明 |
| --- | --- |
| LINK_VALUE | Data Pump は、コードが追加されなくても、マルチプロセッサ・システムでの並列処理をサポートします。並列処理が使用されている場合、このパラメータにより、同じスレッド内で実行する必要のあるトランザクションがグループ化されます。 |
| P_EFFECTIVE_DATE | 給与計算アップロードの有効日 |
| P_CHECK_NUMBER | CHECK_NUMBER 列にアップロードするデータ |
| P_PAYMENT_DATE | PAYMENT_DATE 列にアップロードするデータ |
| P_AMOUNT | AMOUNT 列にアップロードするデータ |
| P_CHECK_TYPE | CHECK_TYPE 列にアップロードするデータ |
| P_PRIORITY | 個人支払方法の解決に役立つ優先順位 |
| P_EMPLOYEE_NUMBER | assignment_id を解決するために使用される従業員の番号 |
| P_PAYROLL_RUN_USER_KEY | PER_BF_PAYROLL_RUNS からの代理 FK |
| P_ORG_PAYMENT_METHOD_NAME | 組織支払方法の名称 |

## バックフィードにより使用されるビジネス・プロセス API

これらの API は、バックフィードの表にデータを挿入するために Data Pump によってコールされます。

| パッケージ名 | 説明 | プロシージャ名 |
|---|---|---|
| PER_BF_PAYROLL_RUNS_API | この API は、PER_BR_PAYROLL_RUNS 表を保守するために使用されます。 | CREATE_PAYROLL_RUN<br>DELETE_PAYROLL_RUN<br>UPDATE_PAYROLL_RUN |
| PER_BF_BALANCE_AMOUNT_API | この API は、PER_BF_BALANCE_AMOUNTS 表および PER_BF_PROCESSED_ASSIGNMENTS 表の情報を保守するために使用されます。アサイメントがすでに給与計算とリンクしている（つまり、PER_BF_PROCESSED_ASSIGNMENTS に行がある）かどうかの確認が API で行われます。行がない場合、バランス情報が格納される前に、行が作成されます。また、DELETE_BALANCE_AMOUNT_API を使用してバランス額情報を表 PER_BF_BALANCE_AMOUNTS から削除するとき、他のバランス額または支払詳細がアサイメントと給与計算との間の結合を使用しているかどうかの確認が行われます。使用していない場合、PER-BF_PROCESSED_ASSIGNMENTS から行が削除されます。 | CREATE_BALANCE_AMOUNT<br>DELETE_BALANCE_AMOUNT<br>UPDATE_BALANCE_AMOUNT |
| PER_BF_BALANCE_TYPES_API | この API は、PER_BR_BALANCE_TYPES 表のデータを保守するために使用されます。 | CREATE_BALANCE_TYPE<br>DELETE_BALANCE_TYPE<br>UPDATE_BALANCE_TYPE |

| パッケージ名 | 説明 | プロシージャ名 |
| --- | --- | --- |
| PER_BF_PAYMENT_DETAILS_API | この API は、PER_BF_PAYMENT_DETAILS 表および PER_BF_PROCESSED_ASSIGNMENTS 表の情報を保守するために使用されます。アサイメントがすでに給与計算とリンクしている（つまり、PER_BF_PROCESSED_ASSIGNMENTS に行がある）かどうかの確認が API で行われます。行がない場合、支払情報が格納される前に、行が作成されます。また、DELETE_PAYMENT_DETAIL_API を使用して支払詳細情報を表 PER_BF_PAYMENT_DETAILS から削除するとき、他のバランス額または支払詳細がアサイメントと給与計算との間の結合を使用しているかどうかの確認が行われます。使用していない場合、PER_BF_PROCESSED_ASSIGNMENTS から行が削除されます。 | CREATE_PAYMENT_DETAIL<br>DELETE_PAYMENT_DETAIL<br>UPDATE_PAYMENT_DETAIL |

# 15

# SQL トレース

## SQL トレース

SQL トレース機能は、SQL 文ごとのパフォーマンス情報を提供します。このトレース機能は、セッションまたはインスタンスのどちらに対しても有効にできます。

トレースする各 SQL 文に対して、次のパフォーマンス情報が生成されます。

- SQL 文テキスト
- 解析、実行回数およびフェッチ回数、CPU/ 経過時間、物理的 / 論理的読込みおよび処理された行数
- 最適化された目標
- 解析中のライブラリ・キャッシュでのミス
- SQL 実行時の説明プラン（Oracle 8.1.6+）
- 解析対象のユーザー
- 再帰的 SQL の深さ

トレース機能を有効にすると、SQL トレース機能が無効にされるまでの間、実行された SQL 文のパフォーマンス情報がトレース・ファイルに書き出されます。

---

**注意：** SQL トレースを使用するには、Oracle 8.1.6 および Oracle Applications リリース 11*i* が必要です。

---

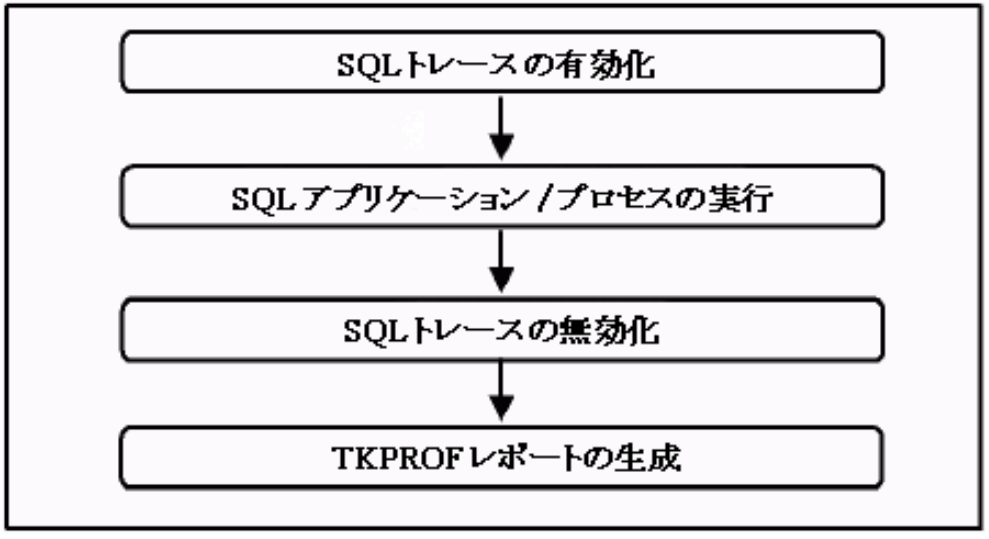
# SQL トレースの使用

SQL トレースを使用するには、最初に SQL トレース機能を有効にしてから、対象となる SQL アプリケーション / プロセス / 文を実行します。すべての SQL 文の実行後、SQL トレースを無効にします。

## トレースの内容の表示

生成されたトレース・ファイルは、オラクル社のレポート用プログラム TKPROF を使用して、ユーザー・フレンドリなレポートに変換できます。または、生成されたトレース・ファイル自体を直接表示することもできます。

**図 15-1 SQL トレースの使用**




**注意：** SQL トレースを有効にすると処理オーバーヘッドが増加しますが、パフォーマンスに与える影響はほとんどありません。

# SQL トレースの有効化

SQL トレースは、init.ora パラメータの sql_trace を使用して有効および無効にします。このパラメータには、TRUE または FALSE のブール値を指定します。このパラメータは、init.ora ファイルでシステム・レベルで設定されます。あるいは、SQL コマンド ALTER SESSION を使用するか、PL/SQL dbms_session.set_sql_trace, dbms_system.set_sql_trace_in_session を使用することで、セッションに対して動的に設定することもできます。

オラクル社が提供するパッケージ・プロシージャがあります。

## 関連するトレースの init.ora パラメータ

次のパラメータは、SQL トレースが有効にされた場合のタイミング、ディレクトリの場所、最大トレース・ファイル・サイズおよびトレース・ファイルのアクセス保護の指定、およびその遵守を可能にします。

| | |
|---|---|
| timed_statistics | 時間統計を収集するかどうかを指定します。有効な値は TRUE または FALSE です。タイミングの分解能は 1/100 秒です。この時間よりも短い操作は、正確に時間がトレースされません。このパラメータが FALSE の場合、タイミングは記録されずにトレース・ファイルに 0 と表示されます。tkprof では、'cpu' 時間および 'elapsed' 時間が 0 になります。 |
| max_dump_file_size | 最大 SQL トレース・ファイル・サイズを指定します。数字のみが指定された場合は O/S ブロック数、K または M が指定された場合はバイト数、UNLIMITED が指定された場合は無限大が指定されます。トレースのサイズが max_dump_file_size を超えた場合は、ファイルの末尾に「*** Trace file full ***」というメッセージが表示されます。 |
| user_dump_dest | SQL トレースが格納されるディレクトリを指定します。値が「?/log」の場合、DBA はデフォルト・トレース・ファイルの格納先を変更しないので、? は $ORACLE_HOME を意味します。 |
| _trace_files_public | トレース・ファイルがパブリック・アクセス設定で書き出されるかどうかを指定します。有効な値は TRUE または FALSE です。 |

## init.ora のパラメータ

これらの init.ora パラメータは、Oracle Session で次の v$parameters 表を確認することで表示できます。

```
SELECT name
      ,value
FROM  v$parameter
WHERE name IN
      ('timed_statistics'
      ,'max_dump_file_size'
      ,'user_dump_dest'
      ,'_trace_file_public');
名称                    値
-----------------      ------
timed_statistics       TRUE
user_dump_dest         usr/oracleHR/log
max_dump_file_size     204800
_trace_file_public     TRUE
```

## SQL トレースの init.ora パラメータの選択

ALTER SESSION コマンドまたは ALTER SYSTEM コマンドを使用すると、timed_statistics および max_dump_file_size をセッション・レベルまたはシステム・レベルで動的に設定できます。

```
ALTER SESSION SET timed_statistics=TRUE;
ALTER SYSTEM SET timed_statistics=FALSE;


ALTER SESSION SET max_dump_file_size=204800;
ALTER SYSTEM SET max_dump_file_size=204800;
```

システム・レベルでは、user_dump_dest パラメータを動的にのみ設定できます。init.ora ファイルでは、_trace_file_public パラメータのみを設定できます。

## Oracle Payroll のプロセスおよびレポートのトレース

SQL トレースが Oracle Payroll に対して有効にされている場合、各プロセスはトレースが実行されたセッションのトレース・ファイルを生成します。「給与計算」など、プロセスがパラレルで実行された場合は、トレース・ファイルがスレッドごとに生成されます。

PAY_ACTION_PARAMETERS 表で TRACE パラメータを設定すると、Oracle Payroll 処理およびレポートを有効または無効にできます。これを行うには、SQL *Plus を使用する方法と「処理パラメータ」ウィンドウを使用する方法の 2 種類があります。

### 方法 1: SQL *Plus の使用

パラメータを**「Y」**（トレース有効）または**「N」**（トレース無効）に設定します。例:

```
/* To enable SQL Trace */
UPDATE    pay_action_parameters
SET       parameter_value = 'Y'
WHERE     parameter_name = 'TRACE';

COMMIT;

/* To disable SQL Trace */
UPDATE    pay_action_parameters
SET       parameter_value = 'N'
WHERE     parameter_name = 'TRACE';

COMMIT;
```

### 方法 2:「処理パラメータ」ウィンドウの使用

「処理パラメータ」ウィンドウを使用してもトレースを有効にできます。

1. パラメータ名として「トレース」を選択します。
2. トレースを有効にするには「Y」を入力し、トレースを無効にするには「N」を入力します。

## 人事管理システム・アプリケーション・フォームのトレース

システム管理者が 'HR Debug Tools' ユーティリティへのアクセスを許可している場合は、人事管理システム・アプリケーション・フォームをトレースできます。

1. メニュー・オプションで「ヘルプ」->「診断メニュー」->「トレース」を選択します。
2. 「トレース」チェック・ボックスをチェックします。

   ユーティリティを無効にするには、「トレース」チェック・ボックスをチェック解除します。

## SQL *Plus からの動的トレース

ALTER SESSION、または PL/SQL パッケージ・プロシージャの dbms_session.set_sql_trace を使用すると、SQL *Plus からトレースできます。どちらの方法を使用しても、SQL_TRACE が有効になっていると SQL 文が実行され、SQL_TRACE が無効になるとトレースが停止されます。

```
SQL> ALTER SESSION SET SQL_TRACE=TRUE;

SQL> Execute SQL statements

SQL> ALTER SESSION SET SQL_TRACE=FALSE;
```

または、

```
SQL> EXECUTE dbms_session.set_sql_trace(TRUE);

SQL> Execute SQL statements

SQL> EXECUTE dbms_session.set_sql_trace(FALSE);
```

dbms_system.set_sql_trace_in_session パッケージ・プロシージャを使用すると、SQL トレース機能を現在有効なすべての Oracle Session で実行できます。このプロシージャは、次の 3 つの引数を受け取ります。

- SID
- SERIAL#
- SQL_TRACE

SID および SERIAL# の値は、v$session 表から判定できます。v$session の列 osuser name、username などをさらにフィルタリングすると、SID/SERIAL# 値を判定しやすくなります。例 :

```
SELECT   s.sid,
         s.serial#
FROM     v$session s
WHERE    s.osuser = 'afergusson'
AND      s.username = 'APPS'


SID     SERIAL#
---     --------
15      19201
execute
dbms_system.set_sql_trace_in_session(15,19201, TRUE);
```

SQL_TRACE 引数はブールで、TRUE または FALSE の値を指定できます。

## トレース・ファイルの検索

トレース・ファイルの場所は、user_dump_dest パラメータを使用して指定します。トレース・ファイル名は、オペレーティング・システムによって異なります。

- UNIX では、トレース・ファイル名が SID_ora_PID.trc です。
- NT では、トレース・ファイル名が oraPID.TRC です。

SID は Oracle システム識別子で、PID はオペレーティング・システムのプロセス識別子です。PID は、v$process および v$session 動的表に有効なセッション ID を問い合せることで判別します。

次の例は、特定の有効なセッションの PID の選択を示しています。

```
SELECT   p.spid,
FROM     v$session s, v$process p
WHERE    s.audsid = &sessionid
AND      p.addr = s.paddr:


SPID
----------
89012
```

## TKPROF とは？

TKPROF は、生成された SQL トレース・ファイルを、ユーザー・フレンドリなファイルに書式設定するプログラムです。このプログラムはトレース・ファイルを読み込み、次のセクションを持つファイルを作成します。

- ヘッダー
- 本文
- 要約

ヘッダー・セクションには、TKPROF のバージョン、使用されたすべての種類のオプション、および用語集が含まれています。本文セクションには、トレースされたユーザー・レベルの SQL 文のそれぞれについて、次の情報が表示されます。

- SQL 文テキスト
- 表形式の解析
- 実行統計およびフェッチ統計
- 解析中のライブラリ・キャッシュのミス数
- 解析ユーザー ID

指定されている場合、TKPROF は次の動作も行います。

- SQL トレースの実行時、および TKPROF の実行時の説明プランの表示
- 表を作成し、各 SQL 文の統計の行を挿入する SQL スクリプトの作成

TKPROF の特長は、SQL 文をソートできることです。ソートを行うことで、多くのリソースを使用する文を識別し、整理しやすくなります。レポートの最後には、すべてのユーザー・レベルおよび再帰的な SQL 文の表形式の要約が用意されています。

## TKPROF を使用したトレース・ファイルの書式設定

TKPROF はコマンド行から実行します。TKPROF には多数の引数がありますが、一般的には 2 つの必須引数および 3 つのオプション引数のみが使用されます。実行構文および引数は次のとおりです。

```
tkprof infile outfile sort=options
explain=username/password@db print=integer
```

### tkprof の引数

tkprof の引数は次のとおりです。

| | |
|---|---|
| infile | 生成された SQL トレース・ファイルを指定します。 |
| outfile | TKPROF のレポート作成先となるファイルを指定します。 |
| explain | 各 SQL 文が説明される、オラクル・ユーザー名、パスワードおよび DB コネクタをオプションで指定します。Oracle 8.1.6 では、説明が指定された場合に 2 つのプランが提供されます。1 つ目のプランは、SQL トレースが生成されたときのためのものです（および、説明オプションの設定に関係なく、常に表示されます）。2 番目のプランは、TKPROF が実行されたときに生成されます。 |
| print | レポートに含める SQL 文の数を制限します。この引数には整数値を指定します。大きな SQL トレース・ファイルの場合に便利です。たとえば、SQL 文を結果の悪いものから 25 個のみ検討する場合は、「print=25」を使用します。 |
| sort | オプションで、ソート順を降順に指定します。順序は 1 つ以上のオプションで指定します。sort 引数を省略した場合、SQL 文はトレース・ファイル内での順序と同じ順序で並べられます。複数のオプションを指定する場合は、それらをスペースで区切ります。使用可能なオプションは次のとおりです。 |

| | |
|---|---|
| PRSCNT | 解析回数 |
| PRSCPU | 解析に要した CPU 時間 |
| PRSELA | 解析に要した経過時間 |
| PRSDSK | 解析中の、ディスクからの物理読込み回数 |
| PRSQRY | 解析中の、整合モード・ブロックの読込み数 |
| PRSCU | 解析中の、カレント・モード・ブロックの読込み数 |
| PRSMIS | 解析中のライブラリ・キャッシュのミス数 |
| EXECNT | 実行回数 |
| EXECPU | 実行に要した CPU 時間 |
| EXEELA | 実行に要した経過時間 |

| | |
|---|---|
| EXEDSK | 実行中の、ディスクからの物理読込み回数 |
| EXEQRY | 実行中の、整合モード・ブロックの読込み数 |
| EXECU | 実行中の、カレント・モード・ブロックの読込み数 |
| EXEROW | 実行中の処理行数 |
| EXEMIS | 実行中のライブラリ・キャッシュのミス数 |
| FCHCNT | フェッチ回数 |
| FCHCPU | フェッチに要した CPU 時間 |
| FCHELA | フェッチに要した経過時間 |
| FCHDSK | フェッチ中の、ディスクからの物理読込み回数 |
| FCHQRY | フェッチ中の、整合モード・ブロックの読込み数 |
| FCHCU | フェッチ中の、カレント・モード・ブロックの読込み数 |
| FCHROW | フェッチした行数 |

**TKPROF の代表的な実行例**

- 標準レポート

```
tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp
```

- 説明オプション付きレポート

```
tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp explain=apps/apps@hrdb
```

- 説明付きで、実行 / フェッチ経過時間でソートした悪い方から 25 個の文のレポート

```
tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp explain=apps/apps@hrdb
sort= exeela fchela print=25
```

## TKPROF のソート・オプション

TKPROF には、トレースした SQL 文のソートを指定できる、多くのソート・オプションが用意されています。お薦めするソート・オプションを次に示します。

- 論理 IO によるソート

  ```
  tkprof infile outfile sort=exeqry execu fchqry fchcu
  ```

- 物理 IO によるソート

  ```
  tkprof infile outfile sort=exeqry execu fchqry fchcu
  ```

- CPU 時間によるソート（timed_statistics が有効な場合のみ）

  ```
  tkprof infile outfile sort=execpu fchcpu prscpu
  ```

- 経過時間によるソート（timed_statistics が有効な場合のみ） tkprof infile outfile sort=exeela fchela prsela

  ライブラリ・キャッシュ・ミスによるソート

- tkprof infile outfile sort=prsmis

HRMS Development では、生成された SQL トレース・ファイルおよび TKPROF レポートの両方を、timed_statistics が TRUE に設定されたときに得られる、実行経過（exeela）時間およびフェッチ経過（fchela）時間でソートすることが適しています。timed_statistics が FALSE の場合は、実行ディスク（exedsk）、実行問合せ（exeqry）および実行 CPU（execu）の各ソート・オプションを使用する必要があります。

# TKPROF レポートについて

tkprof を実行して得られたファイルに含まれるレポートは、次の 3 つのセクションに分かれています。

- ヘッダー
- 本文
- 要約

## TKPROF のヘッダー

ヘッダーには TKPROF のバージョン、実行日、SQL トレース入力ファイル、すべてのソート・オプション（オプションが指定されていない場合は default）、および統計表で使用される用語の用語集が表示されます。

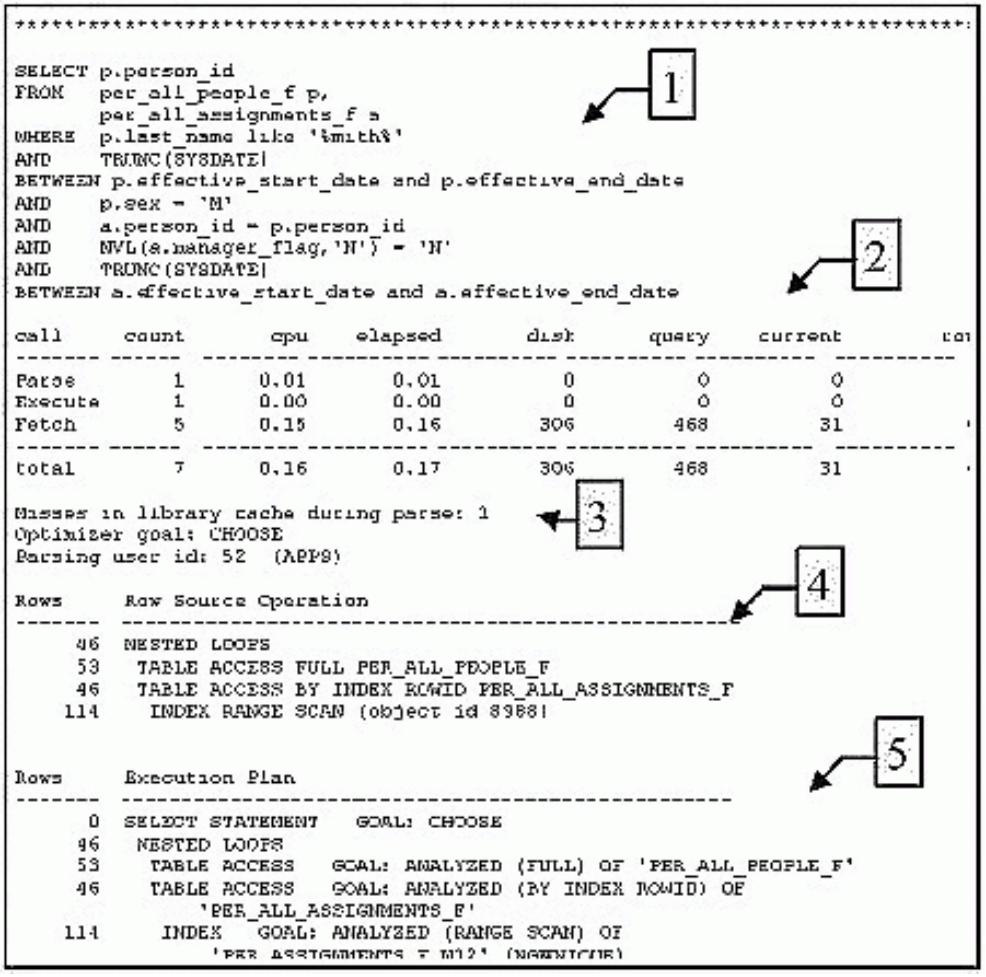
**図 15-2 TKPROF のヘッダー**

```
TKPROF: Release 8.0.6.0.0- Production on Tue Oct 10 04:16:25 2000

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

Trace file: hrqa11i_ora_1211.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************
```

## TKPROF の本文

本文には、トレースされたすべての SQL 文が含まれています。各 SQL 文は、統計および説明プランとともにソート順で表示されます。

**図 15-3 TKPROF の本文**



```
********************************************************************************

SELECT p.person_id
FROM   per_all_people_f p,
       per_all_assignments_f a
WHERE  p.last_name like '%mith%'
AND    TRUNC(SYSDATE)
BETWEEN p.effective_start_date and p.effective_end_date
AND    p.sex = 'M'
AND    a.person_id = p.person_id
AND    NVL(a.manager_flag,'N') = 'N'
AND    TRUNC(SYSDATE)
BETWEEN a.effective_start_date and a.effective_end_date

call     count       cpu    elapsed       disk      query    current        tot
-------  ------  --------  ---------  ----------  ----------  ----------
Parse        1      0.01       0.01           0           0           0
Execute      1      0.00       0.00           0           0           0
Fetch        5      0.15       0.16         306         468          31
-------  ------  --------  ---------  ----------  ----------  ----------
total        7      0.16       0.17         306         468          31

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 52  (APPS)

Rows     Row Source Operation
-------  ---------------------------------------------------
     46  NESTED LOOPS
     53   TABLE ACCESS FULL PER_ALL_PEOPLE_F
     46   TABLE ACCESS BY INDEX ROWID PER_ALL_ASSIGNMENTS_F
    114    INDEX RANGE SCAN (object id 8988)


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
     46   NESTED LOOPS
     53    TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'PER_ALL_PEOPLE_F'
     46    TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF
             'PER_ALL_ASSIGNMENTS_F'
    114     INDEX   GOAL: ANALYZED (RANGE SCAN) OF
              'PER_ASSIGNMENTS_F_N12' (NON-UNIQUE)
```



**1. トレースされた SQL 文が示されます。**

処理中の SQL 文が、バインドされたすべての変数とともに切詰めなしで表示されます。次の SQL 文のみは 25 文字まで切り詰められます。

SET ROLE、GRANT、ALTER USER、ALTER ROLE、CREATE USER、CREATE ROLE

**2. SQL 文の解析、実行およびフェッチの表形式統計が示されます。**

タブ区切りの統計表は、parse、execute および fetch の各コールを検討するための最も重要な情報です。

| | |
|---|---|
| Parse | parse コールは、構文 / 意味チェック、実行プラン生成、および共有カーソルの構築を担当します。解析される SQL 文に応じて、ハード解析またはソフト解析が実行されます。共有キャッシュに SQL 文が見つからない場合は、ハード解析が実行されます。ハード解析では、必要なすべての解析ステップが実行され、最も負荷の高い操作です。共有キャッシュに SQL 文が存在しない場合は、ソフト解析と呼ばれる共有カーソル定義が使用されるため、完全な解析操作が実行される必要はありません。 |
| Execute | SQL 文を実行します。または、SELECT 文が実行されるとフェッチを準備します。 |
| Fetch | SELECT SQL 文から返された行をフェッチします。ORDER BY 句または FOR UPDATE 句が含まれた SELECT 文では、実行中に行がアクセスされます。 |
| count | 各コール・タイプごとのコール数です。 |
| cpu | CPU 時間を表す秒数です（timed_statistics が FALSE の場合は必ず 0)。解析では、共有プールに文が見つかった場合（ライブラリ・キャッシュ・ミスがない場合)、この値は 0 になります。 |
| elapsed | 経過時間を表す秒数です（timed_statistics が FALSE の場合は必ず 0)。 |
| disk | データベース・ファイルからのバッファの物理読込み回数です。（物理 I/O） |
| query | メモリーから整合（問合せ）モードで取得したバッファ数です。（論理 I/O）通常、この列は SELECT 文の処理を反映したものになります。 |
| current | メモリーからカレント・モードで読み込んだバッファ数です。通常、この列は DML INSERT 文、UPDATE 文または DELETE 文の処理を反映したものになります。 |
| rows | 各タイプごとの処理された行数です。SELECT 文の場合、行数は Fetch 列に表示されます。INSERT 文、UPDATE 文および DELETE 文の場合、行数は Execute 列に表示されます。 |

統計情報は、他の統計値および実際の問題が発生している場所へのポインタを判定するために役立ちます。

例 :

合計論理 IO バッファの取得

合計論理 IO = 問合せ合計 + 現在の合計

この統計情報では、メモリーから取得されたデータ・バッファの合計数が得られます。

行ごとの論理 IO

行ごとの論理 IO = 合計論理 IO / 合計行数

この統計情報では、処理された行ごとにメモリーから取得されたデータ・バッファの合計数が得られます。実行された論理 IO の数が多いほど、その行は負荷が高いことになります。理想的には、この比率を可能な限り低くする必要があります。

実行ごとの論理 IO

実行ごとの論理 IO = 合計論理 IO / 実行回数

この統計情報は「行ごとの論理 IO」に似ていますが、実行ごとである点が異なります。

実行ごとの解析回数

実行ごとの解析回数 = 解析回数 / 実行回数

この統計情報は、実行ごとの解析回数を判定します。この値が 1 に近いか等しい場合（1 回以上の実行が行われている場合）、実行のたびに解析が行われ、カーソルが再使用されていません。共有プール・サイズが十分に大きくないため、拡大する必要があります。

バッファ・キャッシュ・ミス率

バッファ・キャッシュ・ミス率 = ディスク合計 / 合計論理 IO

この統計情報では、メモリー内にキャッシュされていないデータのミス率が得られます。理想的には、この数字が 10% 未満になる必要があります。

実行ごとの平均時間

実行ごとの平均時間 = 合計経過時間 / 実行回数

この統計情報では、文の実行に要した平均時間が得られます。この数字は、エンド・ユーザーが受入れ可能な状態かどうかを判定するための、実際的なガイドラインになります。

フェッチごとの平均行数

フェッチごとの平均行数 = フェッチ行数 / フェッチ回数

この統計情報では、fetch コールごとにフェッチされた平均行数が得られます。配列フェッチが使用されているかどうかを判定するために非常に役立ちます。

**3. オプティマイザ・モードが使用され、ユーザー ID を解析しているときの、解析中のライブラリ・キャッシュでのミス数を示します。**

統計情報「Misses in library cache during parse」は、SQL 文でハード解析とソフト解析のどちらが行われたかを示します。ミスが発生した場合（>0）は、SQL 文が共有カーソル・キャッシュ内に見つからず、ハード解析されました。ミスが発生しなかった場合（=0）は、SQL 文が共有カーソル・キャッシュ内に見つかり、ソフト解析されました。この統計情報が常に設定されている（>0）場合、SQL が共有されない理由を調査する必要があります。

統計情報「Optimizer goal」には、オプティマイザが SQL 文の処理に使用した目標が示されます。目標は、次のいずれかの値になります。

`CHOOSE`、`FIRST_ROWS`、`ALL_ROWS` または `RULE`

「Parsing user id」は、SQL コマンドを発行したユーザーを示します。

**4. SQL 文がトレースされると、説明プランが生成されます。**

SQL 文が実行されると、ランタイム説明プランが生成されます。tkprof の引数で説明オプションが指定されたかどうかにかかわらず、この説明プランは必ず表示されます（ただし、ユーザーがすべての基礎オブジェクトにアクセスできない場合は、表示されないこともあります）。また、このプランには参照先のオブジェクトの名称ではなく、オブジェクト ID が含まれます。このようなオブジェクト ID は、object_name を取り出すことのできる all/dba/user_objects 表に直接マップされます。

**5. 説明引数が指定された状態で SQL 文が TKPROF で処理されると、説明プランが生成されます。**

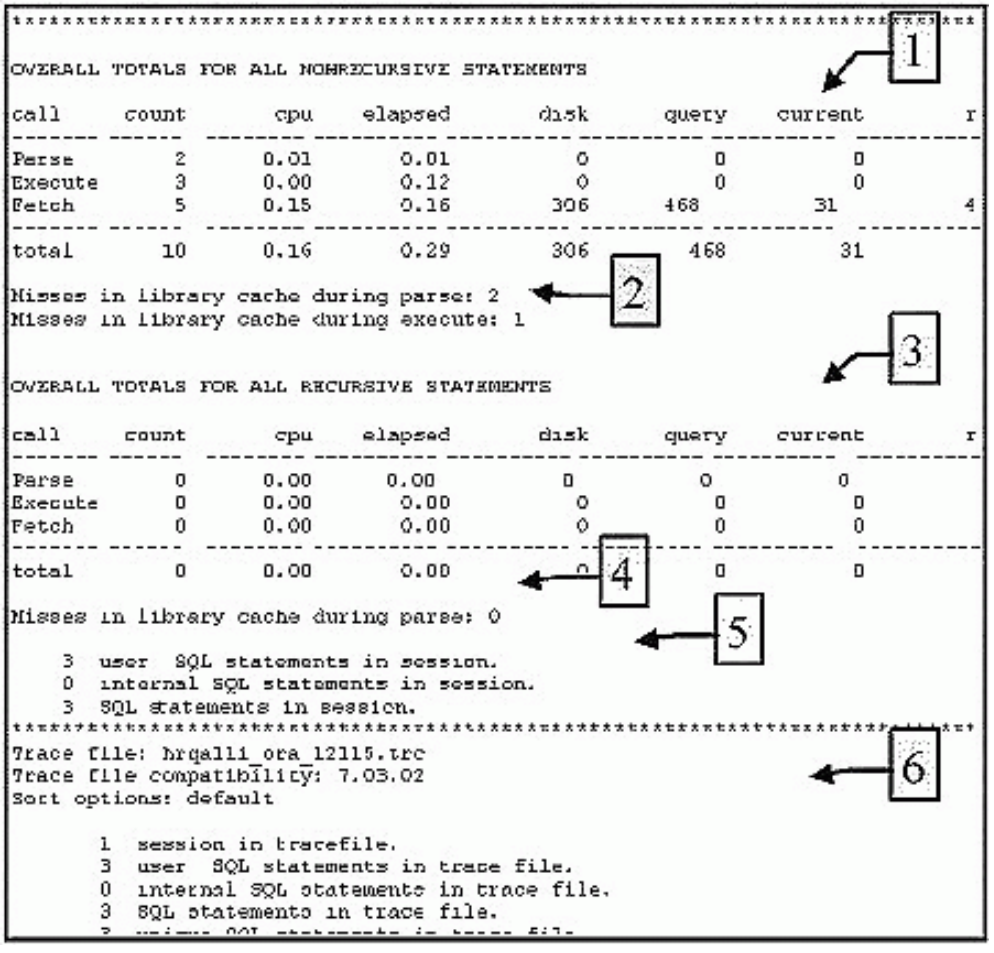
TKPROF で説明引数を指定すると、TKPROF の処理中に各 SQL 文が説明されます。この説明と、SQL トレースの実行時に生成された説明プランとの基本的な違いは、（原価ベースのオプティマイザを使用して）基礎オブジェクトまたは対応するデータベース統計が変更されると、これらの間に差異が生じる可能性があることです。また、すべてオブジェクト ID ではなくオブジェクト名が表示されます。

'Rows' 列には、操作ごとに処理された行数が表示されます。

## TKPROF の要約

要約は、すべてのトレースされた SQL 文よりも後ろの、TKPROF ファイルの末尾にあります。

**図 15-4 TKPROF の要約**



**1. 非再帰的 SQL 文の総合計を示します。**
非再帰的 SQL 文は、開発者が記述した SQL などのユーザー・レベルの SQL 文です。

'OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS' というタブ区切りの表には、発行された再帰的 SQL を除く、ユーザーが発行したすべての文の合計が含まれています（再帰的 SQL の説明は下記の 3 を参照してください）。

**2. 実行および解析中のライブラリ・キャッシュ・ミスを示します。**
前述の本文のセクションで説明したとおりライブラリ・キャッシュ・ミスは、ユーザーなどの共有されていない非再帰的 SQL 文の数を示しています。

**3. 再帰的 SQL 文の総合計を示します。**
再帰的 SQL は、ユーザー SQL 文を完了するために Oracle が発行した内部文です。代表例としては、動的空間管理、紛失したデータ・ディクショナリ情報などがあります。

再帰的 SQL の統計情報は、コールを発行した SQL 文の統計情報には含まれていません。したがって、SQL 文の合計リソース / 原価は、SQL 文と対応するすべての再帰的 SQL 値を加算したものになります。

'OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS' というタブ区切りの表には、SQL トレース・ファイル内のすべての再帰的 SQL の合計が含まれています。この数字は、ユーザー SQL 文を完了するために、Oracle によって内部でどれだけの追加作業が実行されているかを判定する上で重要です。

**4. 解析中のライブラリ・キャッシュ・ミスを示します。**
前述の本文のセクションで説明したとおりライブラリ・キャッシュ・ミスは、共有されていない再帰的 SQL 文の数を示しています。

**5. 処理された SQL 文の要約を示します。**
'user SQL statements in session'（非再帰的）、'0 internal SQL statements in session'（再帰的）および 'SQL statements in session'（非再帰的 SQL 文 + 再帰的 SQL 文の合計）の概要を示しています。

**6. TKPROF の互換性および処理統計を示します。**
処理された SQL トレース・ファイル、トレース・ファイルの互換性およびソート・オプションのリストです。また、セッションの数、一意の SQL 文および SQL トレース・ファイル内の行数も表示されます。

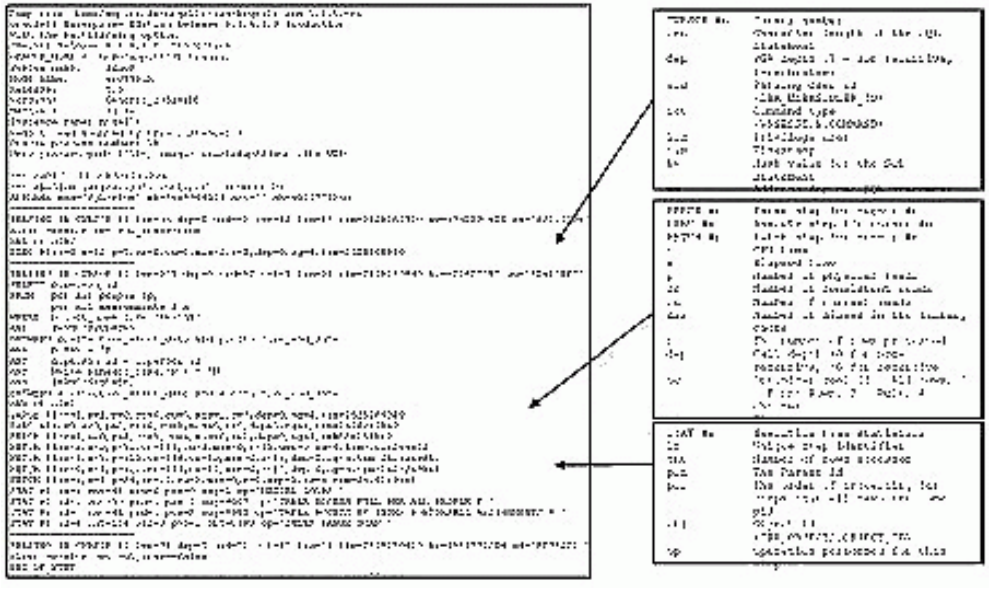
# 生成された SQL トレース・ファイルの例

次の例は、次の 3 つの SQL 文に対して生成された SQL トレース・ファイルのサンプルを示しています。

- ALTER SESSION SET SQL_TRACE=TRUE
- SELECT
- ALTER SESSION SET SQL_TRACE=FALSE.

トレース・ファイルは TKPROF レポートよりも読取りが難しく、ソートもされていません。

**図 15-5 トレース・ファイルの例**



# イベント 10046 を使用した詳細な SQL トレース

10046 イベントを使用すると、バインドされた変数および待機に関する追加情報が、生成された SQL トレース・ファイルでレポートされます。この追加情報は、イベント・レベルの設定で決まります。イベントには、次に示す 4 つのレベル設定があります。

| | |
|---|---|
| 1 | デフォルトの SQL トレース |
| 4 | バインドされた変数情報を含む |
| 8 | 待機イベント情報を含む |
| 12 | バインドされた変数および待機イベント情報を含む |

デフォルトでは、各 SQL トレースがレベル 1 に設定されます。追加情報をレポートするには、ALTER SESSION コマンドを使用して 10046 イベントを必要なレポート・レベルに設定します。

```
ALTER SESSION SET EVENTS '10046 trace name context forever, level 1';
ALTER SESSION SET EVENTS '10046 trace name context forever, level 4';
ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
ALTER SESSION SET EVENTS '10046 trace name context forever, level 12';
```

イベント・レベルを 4、8 または 12 に設定すると、SQL トレースが有効な場合に、生成された SQL トレース・ファイルに追加情報がレポートされます。TKPROF は、イベントの設定によってレポートされた追加情報をすべて無視することに、よく注意してください。

## イベント 10046 バインド変数情報

10046 がレベル 4 または 12 に設定されていると、トレースされる SQL 文にバインド変数が含まれている場合に、バインド変数情報が得られます。使用されているバインド変数値を確認する必要がある場合、この機能は非常に便利です。

**図 15-6 イベント 10046 バインド変数情報**

## イベント 10046 待機イベント情報

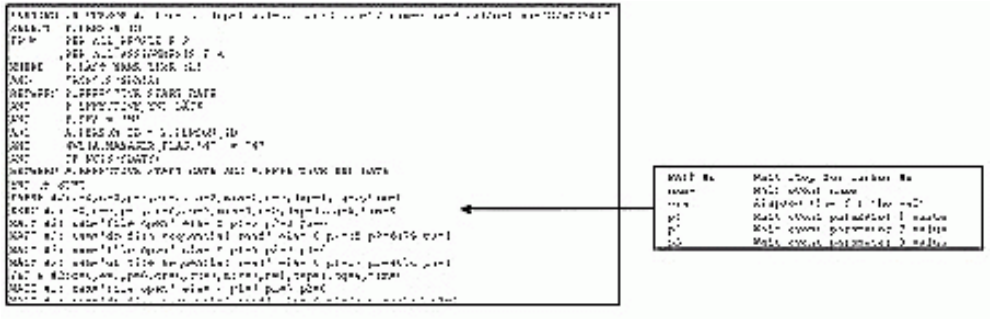
10046 がレベル 8 または 12 に設定されていると、トレースされる SQL 文に待機が含まれている場合に、待機イベント情報が得られます。待機イベント名は、V$SYSTEM_EVENT で得られるイベントと同じです。各イベントには次の 3 つのパラメータがあります。

- p1
- p2
- p3

これらは、V$SESSION_WAIT のパラメータと同じです。イベントおよびパラメータの詳細な説明は、『Oracle 8i リファレンス・ガイド リリース 2 (8.1.6)』を参照してください。

待機イベントのトレースは、SQL 文の経過時間が予定よりも長い理由を調べる場合に非常に役立ちます。たとえば、セッションがラッチ、I/O、SQL*Net などを待っていることがあります。

**図 15-7 イベント 10046 待機イベント情報**



# 用語集

**360 度自己評価（360 Degree Self Appraisal）**

SSHR 評価機能の一部で、グループ評価とも呼ばれる。これは、従業員自身が行う 360 度評価です。従業員（作成者）は評価にマネージャおよびレビュー担当者を加えることができます。

**360 度評価（360 Degree Appraisal）**

SSHR 評価機能の一部で、グループ評価とも呼ばれる。マネージャがレビュー担当者とともに行う従業員評価です。

**BACS**

Banks Automated Clearing System（銀行自動決済システム）の略。これは、従業員の口座に給与を振り込むための UK システムです。

**EIT**

**関連項目：** その他情報タイプ

**ESS**

Employee Self Service（従業員セルフ・サービス）の略。SSHR に事前定義されている職責です。

**LMSS**

Line Manager Self Service（ライン・マネージャ・セルフ・サービス）の略。SSHR に事前定義されている職責です。

**NACHA**

National Automated Clearing House Association（全米自動決済協会）の略。これは、従業員の口座に給与を振り込むための US のシステムです。

**Oracle FastFormula**

プログラミング言語を使用しなくてもオラクル人事管理システムの Formula を作成できる Oracle ツール。

**OSSWA**

Oracle Self Service Web Applications の略。

**OTM**

Oracle Training Management の略。

**Person タイプ（Person Type）**

オラクル人事管理システムには、8 つのシステム定義の Person タイプがあります。このうち 7 つは、従業員、退職者、応募者、前応募者の組合せです。8 つ目のカテゴリは「外部」です。システム定義の Person タイプを基に、ユーザー定義の Person タイプを作成できます。

**QuickPaint レポート（QuickPaint Report）**

従業員や応募者のアサイメント情報についてレポートを作成する方法。まず情報項目を選択し、レポート・レイアウトにペイントします。次にデータ情報を書き加えレポート定義を保存しておくと、いつでも必要なときにレポートが作成できます。

**関連項目：** アサイメント・セット

**QuickPay**

QuickPay を使用すると、1 従業員の給与計算処理を数分で実行できる。従業員が待っているときにその場で計算する場合や、給与計算 Formula をテストする場合に便利です。

**SMP**

**関連項目：** 法定産休手当

**SSHR**

Oracle Self-Service Human Resources の略。イントラネットおよび Web ブラウザを通じて従業員およびそのマネージャに機能を提供する人事管理システム。

**SSP**

**関連項目：** 法定疾病手当

**SSP 資格該当パターン（SSP Qualifying Pattern）**

UK の場合、SSP 資格該当パターンとは、週単位、月単位、または何か他の期間タイプで繰り返される可能性のある資格該当期間。パターンの各週には、資格認定日が最低でも 1 日は含まれている必要があります。資格該当期間は、法定疾病手当（SSP）の支払対象となる唯一の期間であり、ユーザーは、組織の全従業員の SSP 資格該当パターンを定義して、SSP の給付金を計算することができます。

**アクティビティ・レート（Activity Rate）**

医療補償用の従業員給与拠出金として給与期間ごとに $12.35 を支払うなど、アクティビティに関連した金銭の額または率。アクティビティ・レートは加入、適格、補償範囲、拠出金および配分金に適用できます。

### アサイメント（Assignment）

従業員アサイメントは、ビジネス・グループ内での従業員の役割と給与を設定します。アサイメントは複数のアサイメント・コンポーネントから構成されています。その一部である「組織」は、必須のコンポーネントです。また「給与」は支払目的に必要なコンポーネントです。

### アサイメント状況（ステータス）（Assignment Status）

従業員の場合は、退職または一時的な出向、転籍、不就業を追跡して報酬を管理するために使用されるステータス。応募者の場合は採用状況の確認に使用します。

### アサイメント・セット（Assignment Set）

QuickPaint レポートの実行および給与計算処理のためにユーザーが定義する従業員や応募者のグループ。

**関連項目：** QuickPaint レポート

### アサイメント番号（Assignment Number）

従業員アサイメントを一意に識別するための番号。複数のアサイメントを持つ従業員には複数のアサイメント番号が割り当てられます。

### イベント（Event）

従業員または応募者向けに行う、研修、業績考課、会議などのイベント。

### インストラクション（Instructions）

ページの機能を説明するために Web ページに表示される、SSHR のユーザー補助コンポーネント。

### 受取人（Beneficiary）

被保険者の死亡時に福利厚生プランから福利厚生を受け取るために指定された人物または組織。

### 上書き（Overrides）

1 つの給与期間に対してエレメントの支払または入力パラメータを上書きできる。たとえば、給与計算の前に変動処理エレメントのデータ入力のエラーを訂正する場合などに便利です。

### エレメント（Elements）

従業員への支払を処理するコンポーネント。各エレメントには、給与、賃金、持株制度、年金保険料などの報酬または福利厚生タイプがあります。

**エレメント・エントリ（Element Entry）**

従業員ごとにエレメントに入力した値を管理するレコード。入力した値が反映される期間データも含まれます。

**関連項目：** 固定処理エレメント、変動処理エレメント

**エレメント・セット（Element Set）**

給与計算またはカスタマイズしたフォームから、従業員の報酬情報へのアクセスの制御や、配賦原価を定義するためのエレメントのグループ。

**エレメント・リンク（Element Link）**

従業員の複数のアサイメントに対するエレメントの関連付け。エレメントの内容に該当するときにリンクを設定します。アサイメント・コンポーネントとエレメントに該当するリンク・コンポーネントは整合性があります。

**関連項目：** 標準リンク

**オープン登録（Open Enrollment）**

加入者が 1 つまたは複数の福利厚生プランに登録したり、選択を変更できる、予定登録のタイプ。

**応募（Apply for a Job）**

社内公募の必要要員に対して従業員が応募、応募の検索および準備を行うための SSHR の機能。

**応募者（Applicant）**

ビジネス・グループ内の採用応募者。

**応募者内定（Candidate Offers）**

ライン・マネージャが応募者に職務を提示するために使用する SSHR の機能。この機能自体に職責が設定されています。

**階層（Hierarchy）**

レポート・ラインやその他の関連を示す、組織または職階の体系。レポート作成またはオラクル人事管理システム情報へのアクセスの管理に階層を使用できます。

**開発者用付加フレックスフィールド（Developer Descriptive Flexfield）**

国別仕様およびレポートを作成するための条件に従ってローカライズ・チームが定義するフレックスフィールド。

**関連項目：** その他情報タイプ

**カフェテリア福利厚生プラン（Cafeteria Benefits Plan）**

**関連項目：** フレキシブル福利厚生プログラム

**カレンダ（Calendars）**

Oracle Human Resources では、予算年度、四半期、その他の期間の開始日と最終日を設定したカレンダを定義します。カレンダごとに、基本となる期間タイプを選択します。Oracle SSP/SMP では、ユーザーが SSP 資格該当パターンの開始日時を設定したカレンダを定義します。

**カレンダ例外（Calendar Exceptions）**

Oracle SSP/SMP では、ユーザーが SSP 資格該当パターンのカレンダ例外を定義して、指定日のパターンを上書きします。各カレンダ例外は、通常のパターンを上書きする別のパターンです。

**管理登録（Administrative Enrollment）**

プラン条件の変更が原因で再登録する予定登録。

**キー・フレックスフィールド（Key Flexfield）**

複数のセグメントを作成できるフレキシブル・データ・フィールド。各セグメントは、ユーザーが定義するセグメント名と指定した有効値で構成されます。役職、職階、等級、原価コード、従業員グループを一意に識別するために使用します。

**関連項目：** 付加フレックスフィールド

**期間タイプ（Period Type）**

予算計画カレンダにおける、週、月、四半期などの期間区分。

**基本通貨（Base Currency）**

Oracle Payroll で実行する給与計算に使用する通貨。Oracle Payroll は、異なる通貨で従業員に支払が発生した場合、定義された換算レートに基づいて金額を計算します。

**キャリア・パス（Career Path）**

ビジネス・グループ内である役職または職階から次の役職または職階への潜在的な昇進・昇格を示す。キャリア・パスは、役職または職階の昇進・昇格のいずれかに基づく必要があり、混在はできません。

**給与（Payroll）**

Oracle Payroll では、週次、月次、半月単位などの同じ頻度で処理する従業員を分類して実行します。ビジネス・グループでは、必要な数だけ給与計算を設定できます。

**給与計算（Payroll Run）**

すべての給与計算を実行する処理。給与計算の実行間隔は自由に設定できます。

**給与計算項目（Run Item）**

給与計算実行中に、特定のエレメントの処理が支払またはバランスに影響を与える度合。給与計算項目は計算済支払とも呼ばれます。

**給与計算リバーサル（Payroll Reversal）**

給与計算のリバーサルは、1 人の従業員の給与計算のリバーサルのために行う。結果としてその従業員の給与計算は取り消されることになります。

**給与計算ロールバック（Payroll Rollback）**

給与計算全体のリバーサルにより給与計算で処理されたすべての情報を取り消す必要がある場合は、給与計算ロールバックをスケジュールできる。データの整合性を保つために、ロールバックできる給与計算は一度に 1 つのみで、最新の給与計算から開始する必要があります。

**給与振込（Direct Deposit）**

従業員が指定した口座に差引支給額を直接に振り込むための電子転送。

**給与ベース（Salary Basis）**

従業員の給与設定の基本となる時給や年俸などの単位。給与ベースの割当てと受け取る給与エレメントが同じ従業員のグループを定義します。

**業績（査定内）（Performance（Within Assessment））**

一定期間にわたるコンピテンスの「通常」考課の期待内容。たとえば、ある人物はコミュニケーション・コンピテンスにおける業績期待内容を超える場合があります。

**関連項目：** 熟達度（査定内）、コンピテンス、査定

**拠出金（Contribution）**

金銭またはその他の形態で事業主または従業員が福利厚生プランに対して行う拠出。

**勤務選択（Work Choices）**

勤務の優先事項、配属要因または勤務要因とも呼ばれる。これらは、たとえば出張や転勤を希望しているなど、社内の異動の際に影響を及ぼします。勤務選択は、役職および職階の両方または個人レベルで持つことができます。

**国別識別子（National Identifier）**

国内の個人を一意に識別するために使用されるコード。これは課税目的で使用されることがあります。たとえば、US では社会保障番号、イタリアでは会計コード、ニュージーランドでは IRD 番号です。

### クラスに登録（Enroll in a Class）

従業員が社内公開のクラスを検索し、クラスに登録するための SSHR 機能。この機能を使用して、従業員は自分自身のコンピテンスを管理することもできます。

### 繰越（Carry Over）

従業員がある有給休暇期間から次の有給休暇期間に繰り越すことが認められる未使用の有給休暇残余日数。有効期限を設定すれば、特定の日付までに使用されなかった繰越分を取消にすることができます。

**関連項目：** 繰越不可

### 繰越不可（Residual）

有給休暇期間の終わりに従業員が取得資格を失う、未使用の有給休暇残余日数。一般的に従業員は最大日数まで未使用の時間を繰り越すことができますが、この限度を超える繰越分は失われます。

**関連項目：** 繰越

### グループ（Group）

「People グループ」キー・フレックスフィールドを使用して定義するコンポーネント。年金制度や組合など特定のグループに従業員を割り当てるときに使用します。People グループを使用することで、特定のエレメントに対する従業員の適格や給与計算へのアクセスを制限できます。

### グローバル値（Global Value）

Formula に定義して使用する値。グローバル値に設定できるのは、日付、数値、テキストのいずれかです。

### 契約（Contract）

雇用契約とは、雇用する組織と、雇用によりサービスを提供して報酬を受け取る個人との基本的な法的関係を定義する、事業主と従業員または従業員候補との間の契約。雇用契約では、現地の法律に基づき、双方が合意した条件を定義します。

### 結果値（Pay Value）

計算式なしに給与計算項目となるエレメントについてユーザーが入力する値。

**関連項目：** 入力パラメータ

### 原価計算（Costing）

会計およびレポート作成の目的でアサイメント原価を入力すること。Oracle Payroll を使用して計算した原価とその情報を、GL、プロジェクト管理、労務分配などのシステムに転送できます。

**原価タイプ（Costable Type）**

会計および原価計算のために受け取ったエレメントの処理を決定する機能。オラクル人事管理システムには、「原価」、「分配原価」、「固定原価」、「非原価」の 4 つの原価タイプがあります。

**コア・コンピテンス（Core Competencies）**

リーダーシップ・コンピテンスまたは管理コンピテンスとも呼ばれる。会社が目標を達成する上ですべての人に求められるコンピテンスです。

**関連項目：** コンピテンス

**交渉団体（Bargaining Unit）**

事業主または雇用団体に対してあらゆる条件面で交渉する権限を持つ法的に組織化されたグループ。通常は、労働組合または労働組合の 1 支部が交渉団体となります。

**構成可能フォーム（Configurable Forms）**

使用を簡単にするために、またはセキュリティの目的のために、システム管理者がカスタム・フォーム制限によって変更できるフォーム。「フォーム・カスタマイゼーション」ウィンドウには、フォームとその構成方法がリストされます。

**行動インディケータ（Behavioral Indicators）**

勤務においてどのようにコンピテンスが発揮されるかを示す特徴。

**関連項目：** 熟達度レベル

**後任計画（Succession Planning）**

マネージャが後任計画を準備するための SSHR 機能。

**個人検索（Person Search）**

マネージャが個人を検索するための SSHR 機能。単純検索および拡張検索という 2 つのタイプがあります。

**個人情報リスト（People List）**

従業員の検索に使用される、SSHR ライン・マネージャ・ユーティリティ。

**固定処理エレメント（Recurring Elements）**

あらかじめ設定した頻度で定期的に処理するエレメント。固定処理エレメントの値は、削除するまで、または従業員があるエレメントの適用外になるまで保持されます。固定処理エレメントには標準リンクが設定できます。

**関連項目：** 変動処理エレメント、標準リンク

**履用カテゴリ（Employment Category）**

従業員アサイメントのコンポーネント。「フルタイム常勤」、「フルタイム臨時」、「パートタイム常勤」、「パートタイム臨時」の 4 つのカテゴリが定義されています。

**コンピテンス（Competence）**

組織、役職または職階で必要とされる測定可能な行動で、勤務において発揮されるもの。コンピテンスには、知識、スキル、態度、属性といったものがあります。

**コンピテンス・タイプ（Competence Type）**

関連するコンピテンスのグループ。

**コンピテンス評価（Competence Evaluation）**

従業員の特定の職能を評価する方法。

**コンピテンス・プロファイル（Competence Profile）**

応募者および従業員の実績（たとえば、あるコンピテンスの熟達度）を記録する場所。

**コンピテンス要件（Competence Requirements）**

組織、役職または職階によって要求されるコンピテンス。

**関連項目：** コンピテンス、コア・コンピテンス

**再実行（Retry）**

給与計算またはその他の処理を、Post-Run 処理が実行される前に訂正する方法。元の実行結果を削除して、再度処理を実行します。

**裁判所命令（Court Order）**

従業員の給与から養育費補償または債務を天引きし、その控除合計額を裁判所または地方行政当局に支払うことを、事業主に対して求める裁判所の決定。

**採用活動（Recruitment Activity）**

求人募集のためのイベントやプログラム。新聞の求人広告や就職ガイダンスなどがその例です。採用活動は、活動の内容に応じて分類できます。

**作成者（Initiator）**

SSHR において、360 度評価（従業員または自己）を開始する人物。すべての評価情報を見ることができるのは、作成者と評価対象者のみです。

**査定（Assessment）**

職能を評価するために、1 つまたは複数のソースから情報を収集する作業。

**関連項目：** 評価

**産休手当期間（Maternity Pay Period）**

UK の場合、法定産休手当（SMP）が支払われる期間。予定出産日前の第 11 週の初め以降であれば任意の時点から開始し、最高 18 週間まで継続できます。開始日は、通常、従業員との話し合いで決められますが、出産までの任意の時点から開始できます。従業員は、就労している週に関しては、もしくは SMP に関する法律によって定義されている何か他の不適格理由がある場合には、SMP の受給資格はありません。

**参照タイプ（Lookup Types）**

国籍、住所タイプ、税区分など、特定情報の項目に関して有効な値をリスト化した情報のカテゴリ。独自の参照タイプを定義したり、システム定義の参照タイプに値を追加できます。

**資格該当パターン（Qualifying Pattern）**

**関連項目：** SSP 資格該当パターン

**資格タイプ（Qualification Type）**

取得項目、教育資格、免許または試験など、コンピテンスの熟達度を確認する資格方法。

**関連項目：** コンピテンス

**支給スケール（Pay Scale）**

1 つまたは複数の支払レートに関連付ける支給ポイント。従業員の等級または職務経験に準じて、対応するスケールのポイントを決定します。

**関連項目：** 等級スケール

**支給ポイント（Progression Point）**

支給スケールは、支給ポイントを基準に決定します。支給ポイントは、支給スケールに準じて従業員の昇給順序を示します。

**関連項目：** 支給スケール

**自己評価（Self Appraisal）**

SSHR 評価機能の一部。従業員が自分の業績およびコンピテンスを採点するために行う評価です。

**事前定義コンポーネント（Predefined Components）**

一部のエレメントとバランス、すべての第 1 分類と一部の第 2 分類は、国別の法定要件を満たすように Oracle Payroll であらかじめ定義され、製品に組み込まれています。これらの事前定義コンポーネントは削除できません。

**実掛金（Actual Premium）**

特定の福利厚生について保険会社がプラン主催者に請求する、加入者単位の掛金。

**質問表（Questionnaire）**

評価の結果を記録する SSHR 機能。

**自と至（Date To and Date From）**

このフィールドは、デートトラックの対象でないウィンドウで使用します。入力された期間は、自または至のどちらかのフィールドの値を変更するまで変わりません。

**関連項目：** デートトラック、有効日

**支払タイプ（Payment Type）**

従業員への支払方法は、「銀行振込」、「現金」から選択して設定する。このタイプに対応する独自の支払方法を定義できます。

**従業員履歴（Employee Histories）**

従業員が研修履歴、応募履歴、雇用履歴、不就業履歴または給与履歴を表示するための SSHR 機能。マネージャもこの機能を使用して直属の部下の情報を表示できます。

**熟達度（査定内）（Profiency（Within Assessment））**

査定担当者が判断する、一定期間にわたるコンピテンス内専門知識の認知レベル。たとえば、ある人物はエキスパート・レベルのコミュニケーション・コンピテンスを示す場合があります。

**関連項目：** 業績（査定内）、コンピテンス、査定

**熟達度レベル（Proficiency Level）**

勤務においてどのようにコンピテンスが発揮されるかを表現、測定するためのシステム。

**関連項目：** 行動インディケータ

**上限値（Ceiling）**

有給休暇付与プランで従業員が保持可能な有給休暇の残余日数の最大値。最大値に達した場合、従業員は累積時間を消化しないかぎり新しく時間を累積することはできません。

**証明（Certification）**

プラン加入の放棄、扶養家族の補償範囲の指定、または FSA に基づく物品サービスの精算金の受取りなど、ライフ・イベントの結果として、福利厚生プランを新しく選択したり変更するために必要となる文書。

**職階（Position）**

組織と役職から定義する特定のロール。たとえば、「出荷管理」という組織と「事務」という役職からは、「出荷管理事務」という職階を設定できます。

**職責（Responsibility）**

アプリケーションの許可レベル。各職責から Oracle Applications の特定のフォーム、メニュー、レポート、データにアクセスできます。複数のユーザーが 1 つの職責を共有したり、1 人のユーザーが複数の職責を持つことができます。

> **関連項目：** セキュリティ・プロファイル、ユーザー・プロファイル・オプション、要求グループ、セキュリティ・グループ

**新規則の適用除外（Grandfathered）**

福利厚生管理で使用される用語。プランに変更が発生しても、福利厚生が維持される場合、加入者の福利厚生は新規則の適用除外であるとされます。

**セキュリティ・グループ（Security Group）**

人事管理システムのユーザーは、セキュリティ・グループを使用してビジネス・グループ別にデータを分けることができる。セキュリティ・グループ使用可能のセキュリティのみで使用します。

> **関連項目：** 職責、セキュリティ・プロファイル、ユーザー・プロファイル・オプション

**セキュリティ・グループ使用可能（Security Group Enabled）**

以前は「複数ビジネス・グループ間」職責セキュリティと呼ばれていました。このセキュリティ・モデルではセキュリティ・グループを使用するため、1 つの職責を多くのビジネス・グループに関連付けることができる。

**セキュリティ・プロファイル（Security Profile）**

セキュリティ・プロファイルは、ビジネス・グループ内の組織、職階、従業員および応募者のレコードへのアクセスを制御します。システム管理者は、セキュリティ・プロファイルを使用してユーザーの職責を定義します。

> **関連項目：** 職責

**選択可能（Electability）**

プログラム、プランまたはオプションに適用される適格ルールを満たしている福利厚生加入希望者が、福利厚生を選択できるかどうかを決定する処理。福利厚生に対して適格な加入者でも、福利厚生プランの設計で作成されたルールに基づいて、必ずしも福利厚生を選択可能ではありません。

**専門情報（Professional Information）**

従業員が自分の専門詳細を管理するため、またはライン・マネージャが管理下の従業員の専門詳細を管理するための SSHR 機能。

**組織（Organization）**

従業員のアサイメントに必要不可欠なコンポーネント。ビジネス・グループに必要な数だけ組織を定義できます。組織には、企業内部の部署や、必要に応じて人材派遣会社などの外部組織を作成します。組織階層を構築することで、組織間のレポート・ラインの設定やオラクル人事管理システムへのアクセスを制御できます。

**その他情報タイプ（EIT）（Extra Information Type）**

オラクル人事管理システムの 6 つの主要分野について無制限の数の情報タイプを作成できる、開発者用付加フレックスフィールドのタイプ。ローカライズ・チームもまた国別仕様に従って、EIT を事前定義できます。

**関連項目：** 開発者用付加フレックスフィールド

**退職時処理（Termination Rule）**

退職する従業員のエレメントの処理を終了する時期を指定します。従業員の退職日に終了するか、最終処理日まで処理できる状態にするかを指定できます。

**退職従業員（Terminating Employees）**

従業員が退職すると雇用は終了します。従業員に関する情報はシステムに残して、現在のアサイメントをすべて終了します。

**タスク・フロー（Task Flows）**

関連のある複数のウィンドウをリンクし、ボタン操作で効果的にタスクを実行する機能。システム管理者は、ユーザー・ニーズに合せてタスク・フローを作成できます。

**タブ・リージョン（Tabbed Regions）**

選択肢がリスト表示されるウィンドウの一部。少なくとも 1 つの項目は常に表示されます。目的のリージョンのタブをクリックすると、そのリージョンがスタックの一番上に表示されます。

**通貨間分析（Cash Analysis）**

従業員に対して現金での支払が必要なときに規定する異なる通貨単位名。組合規約により、一定の通貨間分析ルールに従う必要があります。

**データベース・アイテム（Database Item）**

特別プログラミングが付加されたオラクル人事管理システムの情報項目で、Oracle FastFormula が Formula に使用するためにそれを検索し、取り出すことができるようにします。

**デートトラック（DateTrack）**

過去または先日付の有効日を変更する際に、デートトラックを使用することで、新しい有効日に反映する情報の入力と確認が可能です。

**関連項目：** 有効日

**適格（Eligibility）**

福利厚生加入希望者が、プログラム、プランまたはオプションに登録できるかどうかを決定するルールを満たすための処理。福利厚生に対して適格な加入者は、選択可能要件も満たす必要があります。

**適合性照合（Suitability Matching）**

マネージャが個人のコンピテンスを比較および評価するための SSHR 機能。

**等級（Grade）**

従業員アサイメントのコンポーネント。従業員ごとに等級を定義し、従業員の給与や報酬エレメントの値を制御します。

**等級スケール（Grade Scale）**

等級に有効なステップ順序。各ステップは支給スケールの 1 つのポイントに対応しています。等級スケールのポイントに従業員を配置すると、各年度または随時、割り当てられた配置は自動的に増加します。

**関連項目：** 支給スケール

**等級ステップ（Grade Step）**

等級スケールの増分。各等級ステップは、支給スケールの 1 つのポイントに対応しています。

**関連項目：** 等級スケール

**等級内支給率（Grade Comparatio）**

従業員が受け取る報酬額と従業員の等級に応じて設定された中間値との比較。

**等級レート（Grade Rate）**

付与された等級に有効と定義された値、または値の範囲。従業員の報酬の入力を検証するために使用します。

**導出要因（Derived Factor）**

1 つまたは複数の福利厚生に対して加入適格またはアクティビティ・レートを判別する計算に使用される要因（年齢、フルタイム雇用のパーセント、勤続期間、報酬レベルまたは期間内の労働時間）。

### 登録処理のタイプ（Enrollment Action Type）

福利厚生登録または登録抹消に必要な処理。

### 特別給与計算（Special Run）

給与期間内の固定処理エレメントの最初の給与計算は標準給与計算になる。同じ期間内のその後の給与計算は特別給与計算と呼ばれます。固定処理エレメントを定義するときには、特別給与計算処理に対して Yes または No を指定します。

### 特別情報タイプ（Special Information Types）

従業員のスキルなどの情報カテゴリ。「個人特別情報」キー・フレックスフィールドを使用して定義します。

### 入力パラメータ（Input Values）

エレメント情報を保持するために定義する値。Oracle Payroll では、Formula で入力パラメータを処理し、エレメントの実行結果を計算します。1 つのエレメントに 15 個まで入力パラメータが設定できます。

### 年功勤続休暇（Long Service Leave）

特定の事業主での規定期間の勤務または雇用の後にその事業主の従業員に与えられる有給休暇。

### 配属要因（Deployment Factors）

**関連項目：** 勤務選択

### パターン（Pattern）

パターンは、指定された頻度で繰り返される一連の時間単位によって構成されます。Oracle SSP/SMP は、SSP 資格該当パターンを使用して、従業員の法定疾病手当（SSP）の受給権を決定します。

### パターン時間単位（Pattern Time Units）

繰返しパターンを指定する時間単位の順序。各時間単位は、時間数、日数、週数の期間を指定します。

### バランス（Balances）

一般に給与計算で生成される一定期間のプラスまたはマイナスの累計。バランスは、結果値、期間、数字を合計できます。

**関連項目：** 事前定義コンポーネント

### バランス挿入（Balance Feeds）

バランスを挿入するために定義されたエレメントの単位と一致する入力パラメータ。

**バランス調整（Balance Adjustment）**

バランスに対して行う訂正処理。ユーザー定義バランスおよびアサイメント・レベルの定義済バランスのみ調整できます。

**バランス累計期間（Balance Dimension）**

バランスでそのバランス挿入が合計される期間、またはアサイメント / トランザクション・セットの合計を算出する期間。「実行」、「期間」、「四半期」、「年」および「ユーザー」等があります。ユーザー定義バランスのリセット・ポイントは自由に選択できます。

**ビジネス・グループ（Business Group）**

オラクル人事管理システム内の最上位の構成単位。ビジネス・グループは、企業全体に対応させることができる他、子会社や事業部などの主要な部門に対応させることも可能です。オラクル人事管理システムに、ビジネス・グループを個別に構築する必要があります。

**日付別検索（Search by Date）**

個人を入社日、応募日、役職掲示日または研修イベント日により検索を行うための SSHR のサブ機能。

**必要要件（Requisition）**

必要要員を採用する基準となる要件。

**ビューワ（SSHR）（Viewer （SSHR））**

評価に関して表示しか許可されていないユーザー。360 度自己評価における評価担当マネージャまたは従業員は、評価に関して表示のみのアクセス権限を指定できます。

**評価（Appraisal）**

評価とは、従業員の業績が査定され、将来の目標が設定されるプロセスです。

**関連項目：** 査定

**評価対象者（Appraisee）**

評価担当者によって評価される個人。

**評価担当者（Appraiser）**

従業員を評価する人物。通常はマネージャが担当します。

**評価担当マネージャ（Appraising Manager）**

従業員 - マネージャまたは 360 度評価を開始および実行する人物。評価担当マネージャは、評価目的を作成できます。

**標準 HRMS セキュリティ（Standard HRMS Security）**

標準のセキュリティ・モデル。このセキュリティ・モデルでは、別のビジネス・グループを参照する場合、別のユーザーとしてログオンする必要があります。

**標準リンク（Standard Link）**

標準リンクが設定されている固定処理エレメントの入力値は、アサイメント・コンポーネントがそのリンクに該当するすべての従業員に対して自動的に作成されます。

**関連項目：** エレメント・リンク、固定処理エレメント

**ヒント（Tips）**

フィールドに関する情報を提供する、SSHR のユーザー補助コンポーネント。

**フィールド（Field）**

情報を入力、表示、更新、削除するウィンドウ内の表示領域や入力領域。

**関連項目：** ブロック、リージョン

**フォーム（Form）**

機能ごとにあらかじめグループ化されています。一部のウィンドウでは、必要に応じてメニューからコールされ表示されます。フォームは、ブロック、リージョン、フィールドから構成されています。

**関連項目：** ブロック、リージョン、フィールド

**付加フレックスフィールド（Descriptive Flexfield）**

業務に必要な追加情報を取得するためにカスタマイズ可能なフィールド。Oracle Applications では、付加フレックスフィールドを使用してカスタマイズしたフィールドのみ有効です。

**関連項目：** キー・フレックスフィールド

**福利厚生（Benefit）**

従業員の待遇のうち、支払以外の部分。福利厚生の例としては、休暇、事業主負担の医療保険、ストック・オプションがあります。

**関連項目：** エレメント

**不就業タイプ（Absence Types）**

病欠または休暇などの不就業カテゴリ。不就業に関する一連のウィンドウで使用するために定義します。

**物品サービス・タイプ（Goods or Service Type）**

福利厚生プランの主催者が精算対象として承認する物品またはサービスのリスト。

**扶養家族（Dependent）**

福利厚生プランでは、プライマリ加入者との関係が証明されていて、加入者がプランの条件に基づいて補償を受けることを指定する人物です。

**プラン主催者（Plan Sponsor）**

福利厚生プランの資金を提供し、管理する責任がある法的エンティティまたはビジネス。通常は事業主になります。

**プラン設計（Plan Design）**

福利厚生プログラムおよびプランを設定するための機能領域。この処理には適格、使用可能オプション、価格設定、プラン年、協力会社管理者、税金の影響、プラン資産、分配オプション、必要なレポートおよび連絡を管理するルールを定義することが含まれます。

**フレキシブル福利厚生プログラム（Flexible Benefits Program）**

福利厚生プランや補償範囲のレベルを従業員が選択できる福利厚生プログラム。一般的には、従業員には、一定量のフレックス・クレジットまたは金額が与えられ、このフレックス・クレジットを使用して福利厚生プランや補償範囲レベルを購入します。

**フレックス・クレジット（Flex Credit）**

フレキシブル福利厚生プログラムにおける、購買能力の単位。従業員は、一般的には金額で表されるフレックス・クレジットを使用して、福利厚生プランや、これらプラン内の補償範囲のレベルを購入します。

**プログラム外プラン（Not in Program Plan）**

プログラムと別個に定義する福利厚生プラン。

**ブロック（Block）**

特定の業務内容や業務全般の情報で構成される、ウィンドウ内で最大の従属単位。ウィンドウには、少なくとも 1 つのブロックがあります。ブロックにはフィールドやリージョンが含まれます。ブロックは影付きの枠線で囲まれます。ブロック内の入力項目を保存しないと、次のブロックには進めません。

**関連項目：** リージョン、フィールド

**分配（Distribution）**

報酬または福利厚生プランに基づく金銭の支払、または有給休暇。

### 分類（Element Classifications）

エレメントの処理順序と挿入するバランスを制御するための分類。Oracle Payroll では、第 1 分類と一部の第 2 分類はシステム定義されています。その他の第 2 分類は、ユーザー定義できます。

### 変動処理エレメント（Nonrecurring Elements）

従業員に対して新たにエントリを行わない限り、1 回の給与期間の処理のみで終了するエレメント。

**関連項目：** 固定処理エレメント

### 報酬（Compensation）

賃金、給与、賞与を含む従業員への支払。

**関連項目：** エレメント

### 報酬オブジェクト（Compensation Object）

Standard Benefits および Oracle Advanced Benefits では、報酬オブジェクトは適格な加入者に提供される福利厚生プランを定義、分類、および管理するために役立ちます。報酬オブジェクトには、プログラム、プラン・タイプ、プラン、オプション、およびこれらの組合せが含まれます。

### 法定産休手当（Statutory Maternity Pay）

UK では、出産のために休暇をとる女性従業員が法定産休手当（SMP）の法定要件を満たしている場合は、法定産休手当（SMP）が支給されます。

### 法定疾病手当（Statutory Sick Pay）

UK では、病気が理由で 4 日以上の休暇をとる従業員が法定疾病手当（SSP）の法定要件を満たしている場合は、法定疾病手当（SSP）が支給されます。

### マネージャ - 従業員評価（Manager-Employee Appraisal）

SSHR 評価機能の一部。マネージャによる従業員の評価です。しかし、評価担当マネージャがマネージャである必要はありません。

### メニュー（Menus）

業務ニーズに合せて、独自のナビゲーション・メニューを設定できます。

### ユーザー・バランス（User Balances）

ユーザーは、累計期間、バランス挿入などの独自のバランスを作成、更新、削除できます。

**関連項目：** バランス

**ユーザー・プロファイル・オプション（User Profile Options）**

システム管理者やユーザーが条件にあわせてオラクル人事管理システムをカスタマイズするための機能。

**関連項目：** 職責、セキュリティ・プロファイル

**ユーザー補助コンポーネント（User Assistance Components）**

ヒントとインストラクションから構成される SSHR のオンライン・ヘルプ。

**有給休暇（Accrual）**

従業員に認められる休暇の時間で、特定の期間に対して累積されます。

**有給休暇期間（Accrual Term）**

1 年間など、有給休暇が算定される期間。ほとんどの有給休暇付与プランでは、有給休暇期間の終わりに、有給休暇累積日数を繰り越すか、取り消す必要があります。プランによっては、固定した開始日および終了日を設定しない、一定期間のローリング有給休暇期間が使用されることもあります。

**有給休暇残（Net Entitlement）**

有給休暇付与プランにより認められた、任意の時点での従業員の未使用の有給休暇日数。

**有給休暇の残余日数計算（Net Accrual Calculation）**

有給休暇の残余日数を算出するために、プランの累積日数に加算するエレメント・エントリまたは減算するエレメント・エントリを定義するルール。

**有給休暇付与期間（Accrual Band）**

従業員に対して累積される有給休暇の時間数を決定する値の範囲。この値には勤続年数、等級、労働時間およびその他の要因が使用されます。

**有給休暇付与プラン（Accrual Plan）**

**関連項目：** 有給休暇付与プラン

**有給休暇付与プラン（PTO Accrual Plan）**

従業員が有給休暇を累積および取得する資格を得るために登録する福利厚生。プランで許可されている不就業理由、登録可能者、累積日数、使用時期およびその他のルールがプランに定義されます。

**有効日（Effective Date）**

情報の入力日付や表示日付。有効日は、「有効日変更」ウィンドウで設定します。

**関連項目：** デートトラック

### 要求グループ（Request Groups）

特定の職責を持つユーザーが実行できるレポートや処理リスト。

**関連項目：** 職責

### 予算計画値（Budget Value）

Oracle Human Resources では、アサイメントごとに予算計画値と実績を入力することで、組織や職階階層内の実際の要員レベルと計画要員レベルとの差異を評価します。

### 予定登録（Scheduled Enrollment）

オープン登録など、事前定義された登録期間中に行われる福利厚生プランの登録。予定登録には、管理、オープンまたは無制限があります。

### ライフ・イベント（Life Event）

福利厚生に関して適格または不適格の変更に影響を及ぼす個人の様々な変化。

### ライフ・イベント衝突（Life Event Collision）

複数のライフ・イベントの影響により、加入適格、登録可能性、補償範囲のレベルまたはアクティビティ・レートの相互矛盾が発生する状況。

### ライフ・イベント登録（Life Event Enrollment）

プラン年中に発生したライフ・イベントによる福利厚生プランへの登録。

### リージョン（Region）

ウィンドウ内のフィールドに対し、論理的に関連する項目をリスト化したもの。ボックスまたは横線で他のフィールドと区切られています。

**関連項目：** ブロック、フィールド

### リバーサル（Reversal）

給与計算または QuickPay 実行を、Post-Run 処理が行われた後に訂正する方法。プラスの結果値をマイナスの結果値に置換え、マイナスの結果値をプラスの結果値に置換えます。新旧の両方の値がデータベース上に残ります。

### 累積期間（Accrual Period）

有給休暇期間内で有給休暇が累積される期間の単位。多くのプランでは、1 か月に 2 日など、累積期間ごとに同じ時間が累積されます。期間ごとに累積時間が異なるプランもあります。また、有給休暇期間の開始時に、その有給休暇期間全体の時間数が認定されるプランもあります。

**レート（Rates）**

従業員の等級または支給ポイントに対する値。たとえば、給与レートと残業レートを定義します。

**レート・スケール（Rating Scale）**

会社のコンピテンスを一般的な形で表現（評価）するために使用。コンピテンス・レベルでは熟達度レベルを持ちません。

**関連項目：** 熟達度レベル

**レビュー担当者（SSHR）（Reviewer（SSHR））**

評価担当マネージャにより招聘され、評価にレビュー・コメントを追加する個人。

**レポート・セット（Report Set）**

同時処理を実行するように指定したレポートとコンカレント・プロセスのグループ。

**レポート・パラメータ（Report Parameters）**

レポートを実行するときに、並べ替え、書式設定、選択を制御したりレポートの情報を要約するときに入力します。

**連結セット（Consolidation Set）**

同一期間内に実行する複数の給与計算をグループ化し、そのグループに対して、レポート作成、原価計算、Post-Run 処理のスケジュールを作成する機能。

**連絡（Communications）**

なんらかの方法で加入者に対して提示される福利厚生プランの情報。登録説明用パッケージ、登録確認書またはデフォルト登録の通知などがこれにあたります。

**連絡先（Contact）**

登録する従業員の続柄関係にある人物。連絡先には、扶養家族、親戚、友人などの緊急時の連絡先を設定します。

**ロールバック（Rollback）**

給与計算またはその他の処理を、Post-Run 処理が行われる前に削除する方法。すべてのアサイメントと実行結果を削除します。

**労働協約（Collective Agreement）**

事業主、または事業主の団体など事業主の代表者と、労働組合または労働組合の支部などの交渉団体との間の契約形態。

**労働協約等級（Collective Agreement Grade）**

労働協約で、従業員を査定し、または等級付ける方法を決定するための情報の組合せ。

### ワーク・ストラクチャ（Work Structures）

組織、役職、等級、給与およびその他の従業員の分類に関する基本的な定義。これによって、従業員の仕事のアサイメントを定義する基本構造が決まります。

### ワークフロー（Workflow）

チャートを使用して承認プロセスを管理する Oracle のアプリケーション。また、Web ページ内のセクションの表示値および説明を構成するために SSHR でも使用されます。